ISBN 82-553-0358-8

Mathematics
No 12 - September 1978

COMPUTING IN ALGEBRAIC SYSTEMS

by

J.V. Tucker

PREPRINT SERIES - Matematisk institutt, Universitetet i Oslo

COMPUTING IN ALGEBRAIC SYSTEMS

J.V. Tucker

We shall construe computation in a relational structure A through algorithms written for four kinds of theoretical computing device each designed to operate within A. First of all there is the standard A-register machine which can hold in its registers a fixed, finite number of elements of A, perform the basic operations and decide the basic relations on these elements, and manage some simple manipulations and decisions such as to replace the contents of one register by those of another and to tell when two registers carry the same element. Secondly, there is the A-register machine with counting which sees a finite number of counting registers added to an A-register machine; these carry natural numbers and the machine is able to put zero into a counting register, add or subtract one from the contents of any counting register, tell if two registers contain the same number, and so on. Thirdly, there is the A-register machine with stacking which augments an A-register machine with a single stack register into which the entire contents of the ordinary algebraic registers of the basic device can be temporarily placed, as a vector along with one of a finite number of prescribed markers, at various points in the course of a calculation. Thus the combinatorial operations of the A-register machine are extended in the first instance by permitting subcomputations on the natural numbers ω, and in the second by prolonging the number and complexity of entirely algebraic subcomputations. The fourth

and primary device is the A-register machine with
stacking which is an A-register machine with both
and stack adjoined.

To use one of these machines to compute a par-
$f : A^n \to A$ is to write down the familiar finite pr-
tions referring to whatever activities of the mac-
and containing information to stop in certain cir-
given $a \in A^n$ as an input the programme is suppos-
machine until $f(a)$, if it exists, lies in an out-
a programme is called a finite algorithmic proce-
rithmic procedure with counting, a finite algorit-
stacking, or a finite algorithmic procedure with
stacking accordingly as it includes or ignores ir-
ing the counting and stack facilities. These pro-
abbreviated in turn by fap, fapC, fapS and fapC-
functions they compute are the subject of this p-

The idea of the A-register machine first app-
Herman and S.D. Isard's [15] and in H. Friedman's
together with the A-register machine with countir-
stacking mechanism belongs to [25,26] in which J.
V. Stoltenberg-Hansen, and I investigated in som-
theoretic properties of the four classes of fini-
procedures. These two papers of ours contain ne-
for this article which continues the study of ma-
into an algebraic milieu, asking questions perti-
rings and fields rather than to general consider-

Still, one expects this report to interest most those involved with Computation and Recursion Theory: it shows that common algebraic properties of A materially affect, indeed determine, the fine structure of the fapCS-computation theory over A and that this occasions diverse circumstances in computing, quite different from recursion theory on ω, the natural numbers. Such situations merit study and lead to an algebraic explanation of all the familiar recursion-theoretic structure. For example, we learn that what is special about computation on ω, in the way of pairing, selection and the like, derives from the fact that $(\omega;S,P,0,=)$, where S and P are the successor and predecessor functions, is an infinite algebra prime in the sense that it contains no proper subalgebras, see theorems 4.4, 5.8. So any other such algebra enjoys the same advantages, for example the rational number field $\mathbb{Q}$. And because it shows that the fapCS-computable functions are the proper framework in which to conduct complexity of computation arguments about polynomials and so on, indeed to create a general theory.

But it offers more. The paper supports possibly contentious opinions on constructivity and algebra, partially in the form of a Church-Turing Thesis generalised to abstract algebraic systems set down in [27], though something is to be said about this in section two. It does so in content and by envisaging the development of fapCS-computation as an enterprise which should cohere with algebraic thinking, to be used to explicate questions to do with constructiveness, complexity, or definability wherever they arise in Algebra, in short by helping to design the theory as a useful addition to the algebraist's toolkit.

Here is the course of the paper. The first two sections introduce the functions properly and recount the principal results of [25,26,27]. In section three, machine computability is established as an invariant of algebra isomorphisms. Section four deals with algorithms for enumerating subalgebras and initiates a local point of view toward computing - characteristic of fapCS-computation - in two theorems about general complexity measures on the fapCS-computable functions. Section five is a technical account of fapCS-semicomputability. Sections six to eight examine some relationships existing between the basic A-register machine and the counting and stack faculties, particularly in structures such as groups and fields. Lastly, in section nine, we prove a useful theorem about computation in topological algebras from which it is deduced that several simple algebraic relations are fapCS-semicomputable but not in general fapCS-decidable.

The article owes much to J.C. Shepherdson's exegesis of Friedman's paper in Theory of Computation seminars at the University of Bristol. But the final form of the material is influenced most from my collaboration with J. Moldestad and V. Stoltenberg-Hansen in [25,26]. I must also thank other colleagues, J.E. Fenstad and D. Normann, for their ideas on generalised recursion theory explained in many conversations and seminars at Oslo. And I gratefully acknowledge the indispensible support of a research fellowship from the European Programme of the Royal Society, London.

1. Finite algorithmic procedures

This section is not quite all preparatory ideas, their definition and scholia, it does contain some important facts, but let us begin conventions and notation. Throughout we are exclusively concerned with relational structures of the form $A = (A;\sigma_1,\ldots,\sigma_p,R_1,\ldots,R_q)$ where each operation and relation is finitary and each relation, but not necessarily each operation, is total. The word function, unqualified, will mean partial function and typically these will be (n,m)-ary functions either $A^n \times \omega^m \to A$ or $A^n \times \omega^m \to \omega$, a distinction we preserve but abbreviate by $A^n \times \omega^m \to A/\omega$; for $(a,x) \in A^n \times \omega^m$, $f(a,x) \simeq g(a,x)$ means $f(a,x), g(a,x)$ are both defined and equal or are both undefined. On fixing part of the argument $a \in A^n$ of a function $f : A^{n+m} \to A$ we write $f_a(b) \simeq f(a,b)$ and $f(a) : A^m \to A$. On extending a unary function $f : A \to A$ to an n-fold cartesian product $f\times\ldots\times f : A^n \to A^n$ we write for $a = (a_1,\ldots,a_n) \in A^n$ the value $f(a)$ for $(fa_1,\ldots,fa_n)$. Relations are often identified with their characteristic functions thus:

$R(a,x) = 0$ if $R(a,x)$ is true; A^* is the set of all finite sequences of

$= 1$ if $R(a,x)$ is false.

elements of A, and $\neg A$ denotes the complement of A.

$\omega, \mathbb{Z}, \mathbb{Z}_p, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ are the natural numbers, the integers, the integers modulo p, the rationals, reals and complex numbers respectively.

The reader is assumed acquainted with Computability and Recursive Function Theory on ω and, of course, with the elements of Algebra. In the hope of engaging a diverse audience we have made this background elementary; for those ideas and facts left unexplained in the text we recommend the books of M. Davis [4] and A.I. Mal'cev [22] on Computability, those of A.G. Kurosh [19] and Mal'cev [23] on Universal Algebra, that of Kurosh [17,18] on Group Theory and B.L. van der Waerden [32] for Field Theory.

To understand the nature of A-register machines and their auxiliary components is to be precise about their programmes. Programmes for A-register machines are written in this language: variables are $r_0, r_1, r_2, \ldots$ for algebra registers. Function and relational symbols are those used for the species of the relational structure A, and include equality. There is one constant H for halt.

A finite algorithmic procedure, or fap, is a finite ordered list of instructions $I_1, \ldots, I_l$ which are of three kinds. The algebraic operational instructions manipulate elements of A and are

A0.1: $r_\mu := \sigma(r_{\lambda_1}, \ldots, r_{\lambda_k})$ meaning "apply the k-ary operation σ to the contents of registers $r_{\lambda_1}, \ldots, r_{\lambda_k}$ and replace the content of register r_μ by this value". Notice the instruction can be specified by its characteristic numerical parameters $\langle ao\text{-}1, \mu, \sigma, \lambda_1, \ldots, \lambda_k \rangle$ where we let ao-1 stand for a number which refers to the kind of the instruction and σ stand for that i such that σ is σ_i, a point taken up later on in this section when we consider coding.

A0.2: $r_\mu := r_\lambda$ meaning "replace the content of register r_μ with that of r_λ". This has characteristic parameters $\langle ao\text{-}2, \mu, \lambda \rangle$.

The algebraic conditional instructions determine the order of implementing instructions and are

AC.1: if $R(r_{\lambda_1}, \ldots, r_{\lambda_k})$ then u else v meaning "if the k-ary relation R is true of the contents of $r_{\lambda_1}, \ldots, r_{\lambda_k}$ then the next instruction is I_u otherwise it is I_v". This has characteristic parameters $\langle ac\text{-}1, R, \lambda_1, \ldots, \lambda_k, u, v \rangle$ where R stands for the i such that R is R_i.

AC.2: if $r_\mu = r_\lambda$ then u else v meaning "if registers r_μ and r_λ contain the same element then the next instruction is I_u otherwise it is I_v" which is parameterised by <ac-2,μ,λ,u,v>. If $\mu = \lambda$ then the instruction is written goto u.

And, thirdly, H.1: H meaning "stop" and coded <h-1>. These types of instruction we refer to as algebraic or fap instructions.

For the A-register machine with counting the programme language is extended by variables $c_0, c_1, c_2, \ldots$ for counting registers and function symbols, 0 for the zero function, +1 for the successor function, -1 for the predecessor function.

A finite algorithmic procedure with counting, or fapC, is a finite ordered list of instructions which may be fap instructions or counting, or arithmetical, instructions of these kinds: operational:

CO.1: $c_\mu := 0$ meaning "make 0 the content of register c_μ", parameterised by <co-1,μ>.

CO.2: $c_\mu := c_\lambda + 1$ meaning "add 1 to the content of c_λ and replace the content of c_μ with this value"; this has parameters <co-2,μ,λ>.

CO.3: $c_\mu := c_\lambda - 1$ meaning "if the content of c_λ is not 0 then subtract 1 from it and replace the content of c_μ with this value otherwise make 0 the content of c_μ"; this has parameters <co-3,μ,λ>.

And conditional: CC.1: if $c_\mu = c_\lambda$ then u else v meaning "if registers c_μ and c_λ contain the same number then the next instruction is I_u else it is I_v"; this has parameters <cc-1,μ,λ,u,v>.

For the A-register machine with stacking we add to the programming language variable s for stack register and constants 1,2,... for markers and $\emptyset$ for empty. The operational stack instructions are

SO.1: $s:=(i;r_o,\ldots,r_m)$ meaning "along with the marker i, place the contents of $r_o,\ldots,r_m$ as an (m+1)-tuple into the stack register"; this has parameters <so-1,i,m>.

SO.2: restore $(r_o,\ldots,r_{j-1},r_{j+1},\ldots,r_m)$ meaning "remove the last or topmost vector in the stack and replace the contents of $r_o,\ldots,r_{j-1},r_{j+1},\ldots,r_m$ with the corresponding entries of that vector". This has parameters <so-2,j,m>.

There is one conditional, SC.1: If $s=\emptyset$ then u else v meaning "if the stack is empty then the next instruction is I_u otherwise it is I_v", with code <sc-1,u,v>.

But in employing these instructions in programmes we insist on two conventions: first that the operating instructions for stacks appear only in stacking blocks which are sequences of consequent instructions of the following form

$s:= (i;r_o,\ldots,r_m)$

I_1
.
.
.
I_n
goto k

* : $r_j := r_o$

restore $(r_o,\ldots,r_{j-1},r_{j+1},\ldots,r_m)$

The marker i is unique to the block in any programme in which that block appears. The $I_1,\ldots,I_n$ are fap operational instructions referred to as (re-)loading instructions. The instruction I_k has a special rôle in the operation of the block it is called the return instruction of the block and it must be either a fap instruction outside all the blocks in the programme or it is the first instruction of any block in the programme. The instruction (informally) prefixed by an asterisk is called the exit instruction.

And secondly that the halt instruction takes on the form of a block,

I_i	if $s = \emptyset$ then i+1 else i+2
I_{i+1}	H
I_{i+2}	goto (exit instruction of the block whose marker is topmost in the stack)

This halting block we abbreviate if $s=\emptyset$ then H else *.

Now a finite algorithmic procedure with stacking, or fapS, is a finite ordered list of instructions which may be fap instructions, or stack instructions satisfying the block and halt conditions.

A finite algorithmic procedure with counting and stacking, or fapCS, is a finite ordered list of instructions composed of fap instructions, arithmetic instructions, or stack instructions satisfying the block and halt criteria.

A programme α will always involve an initial segment of the register variables say $r_o,\ldots,r_p,c_o,\ldots,c_q$ and if intended to compute $A^n \times \omega^m \to A/\omega$ reserves the first few registers $r_1,\ldots,r_n,c_1,\ldots,c_m$ as input registers and r_o or c_o as output

register, those remaining being known as working registers; the numbers $\langle t,p,q,n,m,a\rangle$ are characteristic parameters of the programme where t says what type of fap it is and a what kind of output is expected. The instructions making up α are executed on a machine in the order they are given except where a conditional instruction directs otherwise. If an instruction cannot be carried out, such as happens when one applies a conditional to empty registers, or when one of the basic functions is undefined, then the machine is said to hang in that state, and in failing to halt can compute no value. Thus, in association with an appropriate machine, a fap and fapS α typically defines a function $A^n \to A$ by loading $a \in A^n$ into input registers $r_1,\ldots,r_n$ and applying α: if the machine halts and the output register r_o is not empty, then the value of the function $\alpha(a)$ is defined to be the element in r_o, otherwise no value of the function on a is defined. In like manner, a fapC and fapCS α typically defines a function $A^n \times \omega^m \to A/\omega$ with $(a,x) \in A^n \times \omega^m$ placed in $r_1,\ldots,r_n,c_1,\ldots,c_m$ and the value extracted from r_o or c_o.

A function $f : A^n \to A$ is fap-computable, or fapS-computable, if there exists a fap, or a fapS, α and a suitable machine so that for each $a \in A^n$, $f(a) \simeq \alpha(a)$. A function $f : A^n \times \omega^m \to A/\omega$ is fapC-computable, or fapCS-computable, if there exists a fapC, or a fapCS, α and a machine so that for each $(a,x) \in A^n \times \omega^m$, $f(a,x) \simeq \alpha(a,x)$. The sets of all such computable functions $A^n \to A$ are denoted ${}^nFAP(A)$, ${}^nFAPS(A)$, ${}^nFAPC(A)$ and ${}^nFAPCS(A)$ respectively and their sets in all arguments over A are FAP(A), FAPS(A), FAPC(A) and FAPCS(A) respectively.

Sets and relations are fapCS-decidable if their characteristic functions are fapCS-computable.

A set $S \subset A^n$, or $S \subset A^n \times \omega^m$, is fap/fapS-semicomputable, or fapC/fapCS-semicomputable if it is the domain of a fap/fapS, or fapC/fapCS, computable function.

For example, if G is a group then the set FinOrd(G) of all elements of G of finite order is fap-semicomputable: let α be this fap with input register r_1, output register r_o and working register r_2:

1. $r_2 := r_1$
2. if $r_2 = 1$ then 5 else 3
3. $r_2 := r_2 \circ r_1$
4. goto 2
5. $r_o := r_2$
6. H.

It is easy to see $g \in \text{FinOrd}(G)$ iff $\alpha(g)\downarrow$; in section nine it is shown that FinOrd(G) is not in general fapCS-decidable.

Our work with these programmes quite often requires us to depart from this official format and allow flexible, informal descriptions of programmes. For example, if $f : A^n \times \omega^m \to A$ is a fapCS-computable function and is ancilliary to a calculation we wish to make then we write the instruction $r_\mu := f(r_{\lambda_1}, \ldots, r_{\lambda_n}, c_{\mu_1}, \ldots, c_{\mu_m})$ to abbreviate the transcription of a fapCS for f consistent with the programme we are constructing; none of our corruptions will be ambiguous.

The register machine as a device specific to ω was the idea of J.C. Shepherdson and H. Sturgis [29] and they proved that register machine computability - by which they meant the use of counting instructions on an ω-register machine - coincided with recursion on ω. With the help of uniform recursive pairing $\langle\ \rangle_n : \omega^n \to \omega$

(a recursive bijection which collapses the cartesian product structure of ω) it turns out that we can compute any recursive function on ω using any A-register machine with 2 counting registers so that the recursive functions on ω are fapC-computable over any structure A. This is enormously important for our computations in A and we use the special properties of ω-pairing, search with the μ-operator - all the time and so we accept the use of the Church-Turing Thesis in the construction of our programmes.

But the constant functions $A \to A$ are not computable here (so, for example, the finite subsets of A are not fapCS-decidable): they belong to the less delicate study of the practically constructive functions about which more is said in the next section. Often we need to compute with some specific constants $a \in A^n$ in which case they are adjoined to the basic operations to give the structure (A,a).

Let α be a fapCS computing $A^n \times \omega^m \to A/\omega$ and involving p algebra registers and q counting registers together with a stack register and l markers. A state description in a machine computation under α is a list

$(k;a_1,\ldots,a_p,x_1,\ldots,x_q;(z_1,a_{11},\ldots,a_{1p}),\ldots,(z_s;a_{s1},\ldots,a_{sp}))$
$\in \omega \times A^p \times \omega^q \times (\omega \times A^p)^*$ where $a_1,\ldots,a_p$ are the contents of the algebra registers, $x_1,\ldots,x_q$ are those of the counting registers; there are s vectors piled in the stack register, $1 \leq z_i \leq l$ is the marker of the i-th element and a_{ij} the element in the j-th register of the i-th stored vector; finally k is the number of the instruction in α which is to be applied to these elements.

Each state description represents a step in the computation and often we want to unfold a computation $\alpha(a,x)$ into all its stages

thus we use $D_i(\alpha,a,x) = (m_i;a_{ij},x_{ij};(z_{ij},a^i_{jk}))$ to denote the i-th step in the calculation of $\alpha(a,x)$. The length $|\alpha,a,x|$ of the computation $\alpha(a,x)$ is by definition the number of steps involved in computing $\alpha(a,x)$.

If $a = (a_1,\ldots,a_n) \in A^n$ then the subalgebra of A generated by $a_1,\ldots,a_n$ we write $\langle a \rangle$.

1.1 THEOREM Let $f : A^n \times \omega^m \to A$ be fapCS-computable. Then for each $(a,x) \in A^n \times \omega^m$ if $f(a,x)\downarrow$ then $f(a,x) \in \langle a \rangle$ and, moreover, each stage of any fapCS-computation of $f(a,x)$ lies within $\langle a \rangle$ and ω.

Proof. Let α be a fapCS which computes f and assume $\alpha(a,x)\downarrow$. We prove by induction on the length of the computation that each algebra element appearing in each state description associated with $\alpha(a,x)$ lies in $\langle a \rangle$. This is certainly true for the first step in the computation for there the algebra registers contain only the input a. Assume it is true for the i-th state description $D_i(\alpha,a,x)$ and consider how $D_{i+1}(\alpha,a,x)$ arises from it. Apparently there are 12 cases corresponding with the 12 different types of instruction m_i might refer to, but observe that where m_i is a numerical or halt instruction, or a conditional instruction of any kind, the algebra registers of $D_i(\alpha,a,x)$ are unchanged in passing to $D_{i+1}(\alpha,a,x)$. Thus there are only the 4 cases of algebraic operational instruction and stack operational instructions to examine.

Let m_i be $r_\mu := \sigma(r_{\lambda_1},\ldots,r_{\lambda_k})$. Then the transformation is given by $a_{i+1,j} = \sigma(a_{i\lambda_1},\ldots,a_{i\lambda_k})$ if $j=\mu$ which obviously lie
$= a_{ij}$ if $j\neq\mu$

in <a> since the a_{ij} do so.

Let m_i be $r_\mu := r_\lambda$. Then $a_{i+1,j} = a_{i\lambda}$ if $j=\mu$ which again
$= a_{ij}$ if $j\neq\mu$

are in <a>.

The cases of the stack operations follow this last instruction where elements assumed in <a> are relocated from the work registers to the stack or vice versa. Q.E.D.

To begin to illustrate the point made in the introduction that quite novel properties of computation theories over A grow out of the basic structure of the algebra A, notice that if <a> is finite then there are only finitely many possible values for fapCS-computable functions defined on a:

A is locally n-finite if every n-generator subalgebra of A is finite. A is locally finite if it is locally n-finite for each n. If A is locally n-finite then all fapCS-computable functions of $\leq$ n arguments are constrained by the order function |<a>| as a bound, which is, incidentally, fapCS-computable (theorem 4.10). In [11], E.S. Golod shows that for each n there is a group G which is locally n-finite but not locally (n+1)-finite thus in these groups the theory of the fapCS-computable functions $\leq$ n arguments is considerably removed from that of functions of > n arguments (cf. theorems 4.11,8.1).

Here are some examples where 1.1 is used to show certain functions are not fapCS-computable.

Let F be a field and consider the radical function $f(n,a) = \sqrt[n]{a}$ which picks out an n-th root of a if one exists in F. f is not in general fapCS-computable over F. To be specific take $F = \mathbb{R}$ and $n=2$: if $f(2,a) = \sqrt{a}$ were fapCS-computable then $f(2,2) = \pm\sqrt{2}$ would lie in $\langle 2\rangle = \mathbb{Q}$ which is impossible. This simple observation appears as follows to an algebraist.

By an n-ary root function is meant a map $f : F^{n+1} \to F$ such that for each $a = (a_0,\ldots,a_n) \in F^{n+1}$ if $a_0+\ldots+a_nX^n \in F[X]$ has a zero in F then $f(a)\downarrow$ and is one such root. Let F have prime subfield P and let it contain a root of a non-zero, non-linear polynimal $a_0+\ldots+a_nX^n \in P[X]$ which is irreducible over P. Then F has no fapCS-computable n-ary root function for if such an f existed then $f(a)\downarrow$ and $f(a) \in \langle a\rangle = P$ which would contradict the irreducibility of the polynomial with coefficients a. Clearly, the problem of determining in what circumstances there exist n-ary root functions which are fapCS-computable over $(F;\sqrt[n]{})$ belongs centrally in the tradition of Galois Theory.

Secondly, consider the Euclidean Algorithm associated with an Euclidean ring R. An Euclidean ring is an integral domain R which has a degree function $\partial : R \to \omega$ such that (i) if $a\neq 0$ and $b\neq 0$ then $\partial(ab) \geq \partial(a)$ and (ii) for any $a\neq 0, b \in R$ there exist $q,r \in R$ such that $b = qa+r$ and either $r=0$ or $\partial(r) < \partial(a)$. Now if q,r could be fapCS-computed from a,b then $q,r \in \langle a,b\rangle$ and in general this is not the case: take $R = \mathbb{Z}[X]$ with ∂ the usual polynomial degree; let $a = X^7, b=X^2$. It is easy to check that the necessary q,r have the form $q=\lambda X^5, r=(1-\lambda)X^7$ for $\lambda \in \mathbb{Z}$ but clearly $\lambda X^5 \notin \langle X^7, X^2\rangle$ which involves powers of the form X^{2p+7q}, for $p,q \in \omega$, only.

From 1.1 we learn that each algebraic entry in each state description is a polynomial function of the input, an observation from which is derived the idea of the syntactic state description.

The set $T[X_1,\ldots,X_n]$ of <u>terms</u> or <u>polynomials</u> in the <u>indeterminates</u> $X_1,\ldots,X_n$ is inductively defined solely by the clauses (i) $X_1,\ldots,X_n$ are terms, (ii) if $t_1,\ldots,t_k$ are terms and σ is a k-ary operation symbol then $\sigma(t_1,\ldots,t_k)$ is a term. <u>Term height</u> $Ht: T[X_1,\ldots,X_n] \to \omega$ is defined inductively by $Ht(X_i) = 0$ $1 \leq i \leq n$; if $t = \sigma(t_1,\ldots,t_k)$ then $Ht(t) = \max(Ht(t_1),\ldots,Ht(t_k)) + 1$.

Recalling the definition of a state description, a <u>syntactic state description</u> is a list of the form $(k;t_1(X),\ldots,t_p(X),x_1,\ldots,x_q;(z_1,t_{11}(X),\ldots,t_{1p}(X)),\ldots,(z_s,t_{s1}(X),\ldots,t_{sp}(X)))$ where $k,x_1,\ldots,x_q \in \omega$, the z_i are markers and what remains are terms in indeterminates $X = X_1,\ldots,X_n$; when we unfold a computation $\alpha(a,x)$ syntactically we obtain the i-th syntactic state description $T_i(\alpha,a,x) = (m_i; t_{ij},x_{ij},(z_{ij},t_{jk}^i))$ which is itself a mapping from $A^n \times \omega^m$ to state descriptions.

In connection with this we define an equivalence relation on $T[X_1,\ldots,X_n]$: let $v: T[X_1,\ldots,X_n] \times A^n \to A$ be the valuation map defined $v(t,a) = t(a)$. For $t,t' \in T[X_1,\ldots,X_n]$,

$$t \equiv_A t' \quad \text{iff for every } a \in A^n,\ v(t,a) = v(t',a).$$

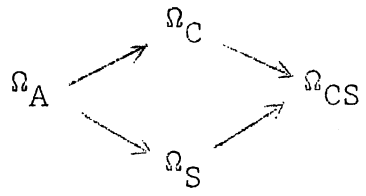
If two syntactic state desceriptions have identical instruction numbers, numerical entries and markers and have terms which are $\equiv_A$ equivalent coordinatewise then those state descriptions can be identified in any fapCS-computation over A.

1.2 REMARK <u>Let $\underline{V}$ be a variety. If $t,t' \in T[X_1,\ldots,X_n]$ and $t = t'$ in $T_{\underline{V}}[X_1,\ldots,X_n]$ the $\underline{V}$-free n-ary polynomial algebra</u>

of $\underline{V}$ then for every algebra A in $\underline{V}$, $t \equiv_A t'$.

These notions are important in section six and, especially, nine.

We conclude with the subject of codings ready for the next section. Each instruction and each programme is essentially a finite string of symbols over the programme language and so the four sets of programmes can be gödel numbered or coded in the usual way. Any coding of these programmes which allows a recursive decomposition into programme parameters and codes for instructions, and from these calculation of the characteristic parameters of the instructions, may be termed a standard coding of the programmes. When formalised such a coding can be shown to be unique up to recursive equivalence in the Mal'cev-Ershov theory of computable numberings, see Mal'cev [21], Ershov [5, 6]. We choose just such codings for each of the four classes of programme along with appropriate recursive embeddings of code sets: Ω_A, Ω_S, Ω_C, Ω_{CS} are the code sets for the sets of fap, fapS, fapC and fapCS programmes respectively and are included



Making numerical coördinatisations of parts of algebras is essential to many of our computations and these are obtained from a coding of polynomials. $T[X_1,\ldots,X_n]$ is assumed numerically coded uniformly in n by a standard coördinatisation ${}^n\gamma_* : {}^n\Omega \subset \omega \to T[X_1,\ldots,X_n]$ in the sense that ${}^n\gamma_*$ is a surjection, ${}^n\Omega$ recursive and there are recursive functions which tell if a code labels an indeterminate and, if it does, which or, if it does not, indicates the leading operational symbol and calculates codes for subterms;

whence term height is recursive $ht:{}^n\Omega \to \omega$. Such a coding is unique up to recursive equivalence in the theory of computable algebras due to Mal'cev [21]. ${}^n\gamma_*$ we abbreviate by ${}^n\gamma_*(i) = [i]$.

1.3 LEMMA *There is a uniform family of recursive functions* ${}^nf:\omega \to {}^n\Omega$ *such that* ${}^n\gamma_*{}^nf:\omega \to T[X_1,\ldots,X_n]$ *is bijective and for* $i,j \in \omega$, $i \leq j$ *implies* $Ht([{}^nf(i)]) \leq Ht([{}^nf(j)])$.

This can be demonstrated from the properties of a standard numbering without much difficulty.

Each term $t(X_1,\ldots,X_n)$ defines a function $A^n \to A$ by substituting algebra elements for indeterminates using v, and we define n-ary term evaluation ${}^nTE:{}^n\Omega \times A^n \to A$ by ${}^nTE(i,a) = [i](a)$.

2. The computable functions in the large

In thinking of the fapCS-computable functions globally we employ the idea of the computation theory which axiomatises the large-scale structure of the partial recursive functions on ω.

A set Θ of functions over A is said to be a *computation theory over* A *with code set* $C \subset A$ and its elements said to be *Θ-computable functions* if associated with Θ is a surjection $\alpha:C \to \Theta$, called a *coding* and abbreviated by $\alpha(e) = \{e\}$ for $e \in C$, and a *length of computation* function $|\ |:C \times A^* \to On$, such that $|e;a|{\downarrow} \iff \{e\}(a){\downarrow}$, for which all the following properties hold.

I. C is acceptable as a code set in that it contains (an isomorpic copy of) ω and Θ contains (functions which correspond to) successor, predessor and zero on ω.

II. Θ contains these generating functions:

(i) for each n and $1 \leq i \leq n$ the projection functions $u_i^n(a_1,\ldots,a_n) = a_i$ with Θ-computable codes $p_1(n,i)$;

(ii) each operation σ of A;

(iii) for each relation R of A, and including the equality relation on A, the definition-by-cases function defined $DC_R(a,x,y) = x$ if $R(a)$, $= y$ if $\neg R(a)$.

III. Θ is uniformly closed under

(i) the composition of functions: if f and g are $n+1$ and n-ary Θ-computable functions with codes $\hat{f},\hat{g}$ respectively their composition defined $C(f,g)(a) \simeq f(g(a),a)$ is Θ-computable with Θ-computable code $p_2(n,\hat{f},\hat{g})$.

(ii) the permuting of arguments: let ${}^j a=(a_j,a_1,..,a_{j-1},a_{j+1},..,a_n)$ when $a = (a_1,...,a_n)$. If f is an n-ary Θ-computable function with code $\hat{f}$ then, for each $1\leq j\leq n$, the function defined ${}^j f(a) \simeq f({}^j a)$ is Θ-computable with Θ-computable code $p_3(n,j,\hat{f})$.

(iii) the addition of arguments: if f is an n-ary Θ-computable function with code $\hat{f}$ then, for any m, the $(n+m)$-ary function g defined $g(a,b) \simeq f(a)$ is Θ-computable with Θ-computable code $p_4(n,m,\hat{f})$.

IV. Θ contains universal functions ${}^n U$ such that for $e \in C, a \in A^n$

$${}^n U(e,a) \simeq \{e\}(a)$$

with Θ-computable code $p_5(n)$.

V. Θ enjoys this iteration property: for each n,m there is a Θ-computable map S^n_m, with Θ-computable code $p_6(n,m)$, such that for $e \in C, a \in C^n, b \in A^m$

$$\{S^n_m(e,a)\}(b) \simeq \{e\}(a,b).$$

And lastly it is required of the length function to respect the efficiency of the functions mentioned in axioms III, IV and V.

VI. (i) Composition: $|p_2(n,\hat{f},\hat{g});a| > \max\{|\hat{f};g(a),a|,|\hat{g};a|\}$

(ii) Permutation: $|p_3(n,j,\hat{f});a| > |\hat{f};{}^{j}a|$.

(iii) Addition: $|p_4(n,m,\hat{f});a| > |\hat{f};a|$.

(iv) Universality: $|p_5(n);e,a| > |e;a|$.

(v) Iteration: $|S_m^n(e,a);b| > |e;a,b|$.

Notice that axiom I ensures a copy of the partial recursive functions on ω is contained within every computation theory.

This definition is from J.E. Fenstad's [7] and improves upon an initiative of Y.N. Moschovakis [28]. In our work here the definition organises the essential structure of the fapCS-computable functions in much the same way that Banach and C^*-algebras are used to handle continuous functions and bounded linear operators, but like these instruments in Analysis, computation theories have accumulated, recently, a theoretical development of their own based upon the traditional resources of recursion on ω, on finite types over ω, and on the ordinals; we refer to the book of J. Moldestad [24] and to V. Stoltenberg-Hansen's [30] and, in particular to the monograph of Fenstad [8], though for the moment the reader need only attend to [26] from which we take these theorems.

The coding of programmes extends to a coding of the functions they compute: choose $C = \omega$ and $\{e\}$ to be the function computed by fapCS e, if $e \in \Omega_{CS}$, or to be the everywhere undefined function otherwise. Define length of computation $|e;a,x|$ to be the number of steps taken in computing $\{e\}(a,x)$, if $\{e\}(a,x)\downarrow$, and to be undefined otherwise. To be faithful to our definition we must now absorb the code set into A by constructing $A_\omega = (A \mathbin{\mathring{\cup}} \omega;\sigma_i,R_i,S,P,Q,\chi_\omega,=)$ in which the operations on ω are undefined on A and similarly those of A undefined on ω, and χ_ω is the characteristic function of ω on $A \mathbin{\mathring{\cup}} \omega$; this has many interesting technical implications immaterial here, see [26] where we prove

2.1 THEOREM The fapCS-computable functions over A constitute a computation theory over A with code set $C = \omega$.

So along with the Universal Computable Functions, and the S-n-m Theorem, the First Recursion Theorem, the Mynill Fixed-point Theorem, are true of fapCS-computation as they are true of any computation theory. We adopt the computation theory as a basic unit of computing power over A because our fapCS-computations requires us to have to hand the full measure of properties guaranteed in the axioms. This is established by experience but one reason is evident from considering the weaker class of fapC-computable functions: repeating the coding conventions for Ω_C and recalling term evaluation,

2.2 THEOREM The fapC-computable functions over A constitute a computation theory over A with code set $C = \omega$ if, and only if, term evaluation is uniformly fapC-computable over A.

The analysis in [26] went further. Let Θ and Φ be computation theories over A with code set C. Then Θ is said to be a sub-computation theory of Φ if $\Theta \subset \Phi$ and there exists a Φ-computable map $p:\omega \times C \to C$ such that for each $e \in C, a \in A^n$, $\{e\}_\Theta(a) \simeq \{p(n,e)\}_\Phi(a)$ and, of course, $|e;a|_\Theta \leq |p(n,e),a|_\Phi$.

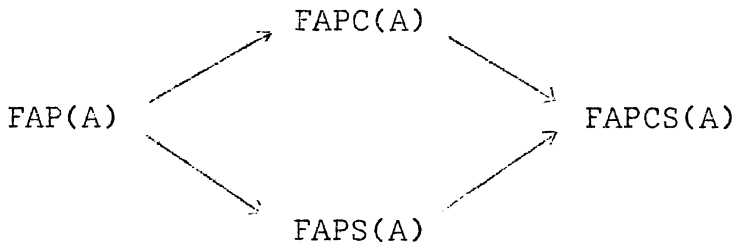
Θ is said to be the minimal computation theory over A with code set C if whenever Φ is a computation theory over A with code set C then Θ is a subcomputation theory of Φ.

2.3 THEOREM The fapCS-computable functions over A constitute the minimal computation over A with code set $C = \omega$.

Actually in groups, rings and fields term evaluation is fapC-computable and so according to 2.2, in all such cases the fapC-computable functions become a subcomputation theory of the fapCS-computable functions. By 2.3 we must have fapC = fapCS.

Nevertheless in [26] it was shown:

2.4 THEOREM There exists a structure A where the following inclusions are strict



And so the notions of counting and stacking are necessary additions both conceptually and from the point of view of 2.3.

Another concept we need is that of the isomorphism of two computation theories. Let Θ,Φ be computation theories over A,B with code sets C,D. A homomorphism $\Theta \to \Phi$ is a pair of maps $\psi : C \to D, \phi : A \to B$ such that ψ is a code homomorphism with respect to successor, predecessor and zero, ϕ is a relational homomorphism and such that the following diagrams commute: for each n,

$$\begin{array}{ccc} C \times A^n & \xrightarrow{{}^nU} & A \\ \psi \downarrow \quad \downarrow \phi & & \downarrow \phi \\ D \times B^n & \xrightarrow{{}^nU} & B \end{array} \qquad \begin{array}{ccc} C \times A^n & \overset{\|_n}{\searrow} & \\ \psi \downarrow \quad \downarrow \phi & & O_n \\ D \times B^n & \underset{\|_n}{\nearrow} & \end{array}$$

which means ${}^nU(\psi(e),\phi a) \simeq \phi U(e,a)$ and $|e;a|_n \simeq |\psi e;\phi a|_n$.

Θ and Φ are isomorphic if there exist homomorphisms which are inverses to oneanother on codes and algebras; this is equivalent to the existence of a homomorphism which is bijective between codes and algebras.

In the next section we prove that our fapCS-computation theories are isomorphism invariants of abstract algebraic systems.

To appreciate the significance of computation theories in our work one must understand the role they play in the search for a definitive definition of computability in an algebraic setting and the subject of the companion paper [27]. This article depends upon [27] because it is prepared with this extra-mathematical hypothesis in mind: to an algebraist, conceptually, the constructive functions directly derived from an algebraic structure A are the fapCS-computable functions over A. And for all practical purposes the constructive functions derived from A and the fapCS-computable functions augmented by all the constant functions on A. And we do believe these two statements satisfactorily resolve the difficulties involved with advancing a Generalised Church-Turing Thesis. The arguments leading to these assertions are made up from conceptual discussions of constructivity and complexity on the one hand and from reflections on the historical development of algebraic thinking upon the other, taken with technical evidence from Logic and Algebra.

The technical evidence from Mathematical Logic is contained in theorems such as 2.1-2.4 and 3.2 here, and equivalence theorems such as that the fapS-computable functions coincide with the inductively definable functions of R.A. Platek, proved in [25], and that they coincide with a natural generalisation to algebraic systems of Herbrand-Gödel-Kleene equational definability, proved in [27]. And that fapS and fapCS coincide with several other disparate points of view of generalised computing such as program schemes, set recursion, and so on, also established in [27]. The technical evidence from Algebra is the use of the fapCS-computable functions in algebraic structures: in the theorems and minutia of this paper.

Consider more carefully the length of computation function $||: C \times A^* \times \omega^* \to \omega$ on fapCS-computations.

2.5 THEOREM For each n,m the (n,m)-ary step counting function ${}^{n,m}S : C \times A^n \times \omega^m \to \omega$ defined ${}^{n,m}S(e,a,x) \simeq |e;a,x|$ is fapCS-computable and, moreover, the relation ${}^{n,m}Step(e,a,x,k)$ iff ${}^{n,m}S(e,a,x) = k$ is fapCS-decidable, and both functions are uniformly prescribable from n,m.

Proof. This is straightforward given the proof of the universal function axiom for fapCS-computation in [26], theorem 4.1. There one must add a counter at the head of the main programme MP which is entered once for each step simulated by a subroutine. Q.E.D.

2.6 THEOREM $S \subset A^n \times \omega^m$ is fapCS-decidable if, and only if, S and $\neg S$ are fapCS-semicomputable.

Proof. First given S as fapCS-computable we make programmes α,β which converge on and only on, S, $\neg S$ respectively. α is

1. $c := S(a,x)$
2. if $c = 0$ then 5 else 3
3. $c := c+1$
4. goto 3
5. $c_o := c,H$

and to obtain β one asks if $c = 1$ in instruction 2.

Conversely, let α,β be fapCS's with $S = dom(\alpha)$ and $\neg S = dom(\beta)$. To decide whether or not $(a,x) \in S$ we need to interleave the computations of $\alpha(a,x)$ and $\beta(a,x)$ so as to discover which is first to halt, this is done by using the step counting theorem 2.5. Let e_1,e_2 code α,β then the characteristic function of S is fapCS-computed by this programme:

1. $c := 1$
2. $n_1 := S(e_1,a,x,c)$
3. $n_2 := S(e_2,a,x,c)$
4. if $n_1 = 0$ then 8 else 5
5. if $n_2 = 0$ then 9 else 6
6. $c := c+1$
7. goto 2
8. $c_o := 0,H$
9. $c_o := 1,H$.

Q.E.D.

Reminded of M. Blum's [3] we define the idea of a general complexity measure on the fapCS-computable functions to be taken up in section four.

The family $\{\Phi_e : e\in C\}$ of (n,m)-ary fapCS-computable functions is an <u>(n,m)-ary fapCS-computable complexity measure</u> on the fapCS-computable functions if for each $e \in C$ and $(a,x) \in A^n \times \omega^m, \Phi_e(a,x)\downarrow$ iff $\{e\}(a,x)\downarrow$ and, secondly, the relation $\Phi(e,a,x,k)$ iff $\Phi_e(a,x) = k$ is fapCS-decidable. By a <u>fapCS-computable complexity measure</u> we mean a uniformly computable family of complexity measures of all arguments. Theorem 2.5 says that step counting is such a complexity measure. Notice inequalities analogous to axiom VI can be deduced for a complexity measure.

Here are further operators acting on the fapCS-computable functions. First there is the <u>least number operator</u> acting over ω^k by means of the standard pairing arrangements:

2.7 LEMMA <u>Let $f : A^n \times \omega^m \times \omega^k \to A/\omega$ be a total fapCS-computable function. Then $g : A^n \times \omega^m \times (A/\omega) \to \omega^k$ defined by $g(a,x,y) \simeq (\mu z \in \omega^k)(f(a,x,z)=y)$ is fapCS-computable and a code for g is fapCS-computable from a code for f.</u>

Proof. This is obvious: consider this programme which computes g.

1. c := 1
2. r := f(a,x,<c>)
3. if r=y then 6 else 4
4. c := c+1
5. goto 2
6. c_o := c, H

where <c> is the k-tuple coded by c. Q.E.D.

Given a set of generators $a \in A^n$ one defines the term height of $b \in \langle a \rangle$ with respect to a as the least height of a term $t(X_1,\ldots,X_n)$ such that $t(a) = b$.

2.8 LEMMA The term height function ${}^nHt : A^n \times A \to \omega$ is uniformly fapCS-computable.

Proof. Define ${}^nHt(a,b) = ht.{}^nf(\mu z \in \omega)({}^nTE({}^nfz,a)=b)$ where $ht : {}^n\Omega \to \omega$ calculates term height from term codes and ${}^nf$ is the special enumeration of 1.3; by 2.7 the function is fapCS-computable. Q.E.D.

Of course f may not be partial in 2.7, not even when recursive on ω. In practice our interest in choosing the least solution of an equation will quite often be an interest in constructively choosing some solution. It is easy to design a constructive choice function for partial equations.

2.9 LEMMA Let $f : A^n \times \omega^m \times \omega^k \to A/\omega$ be fapCS-computable. This choice is uniformly fapCS-computable in a code for such an f:

$$\delta(\hat{f},a,x,y) \simeq \lambda z.\ (\mu\langle z_o,z\rangle\in\omega)(Step(\hat{f},a,x,z,z_o)\&^{n,m}U(\hat{f},a,x,z)=y).$$

This follows from 2.7 as Step and the equality relation are total and fapCS-computable. Interleaving also allows us to prove this definition-by-cases theorem:

2.10 LEMMA Let $g_i : A^n \times \omega^m \to \omega$ and $f_i : A^n \times \omega^m \to A/\omega$, $1 \leq i \leq k$, be fapCS-computable functions. Assume that for each $(a,x) \in A^n \times \omega^m$ if there is $1 \leq i \leq k$ so that $g_i(a,x) = 0$ then there is no more than one such i. Then the function

$$\begin{aligned} f(a,x) &= f_1(a,x) && \text{if } g_1(a,x) = 0 \\ &\quad \vdots && \quad \vdots \\ &= f_k(a,x) && \text{if } g_k(a,x) = 0 \\ &= \uparrow && \text{otherwise} \end{aligned}$$

is fapCS-computable.

3. Computability and algebraic invariance

Many of the aspects of fapCS-computation in Algebra which are generally invariant are consequents of this single theorem.

3.1 THEOREM Let A and B be relational systems and $\phi : A \to B$ a relational embedding. If α is a fapCS of their species then

$$\begin{aligned} \alpha(\phi a,x) &\simeq \phi\alpha(a,x) && \text{if } \alpha : A^n \times \omega^m \to A \\ \alpha(\phi a,x) &\simeq \alpha(a,x) && \text{if } \alpha : A^n \times \omega^m \to \omega. \end{aligned}$$

Proof. We prove a more important, but technical, fact about the state descriptions of the computations $\alpha(a,x)$ and $\alpha(\phi a,x)$: if $D_i(\alpha,a,x) = (m_i, a_{ij}, x_{ij}, (z_j, a^i_{jk}))$ then $D_i(\alpha,\phi a,x) = (m_i, \phi a_{ij}, x_{ij}, (z_j, \phi a^i_{jk}))$. This is done by induction on i; for convenience set $D_i(\alpha,\phi a,x) = (n_i, b_{ij}, y_{ij}, (w_j, b^i_{jk}))$.

The basis $i=1$ is clearly true as the initial states of both computations contain only the inputs. Assume the equation is true at the i-th step and consider how $D_{i+1}(\alpha,\phi a,x)$ arises from $D_i(\alpha,\phi a,x)$. The induction hypothesis is that $n_i = m_i$, $b_{ij} = \phi a_{ij}$, $y_{ij} = x_{ij}$, $w_j = z_j$ and $b^i_{jk} = \phi a^i_{jk}$, in the various appropriate ranges of j. We shall consider 5 of the 12 possible types of instruction numbered m_i.

Let m_i be H. Then there are just i state descriptions to both computations and the identity holds by the induction hypothesis.

Let m_i be $r_\mu := \sigma(r_{\lambda_1},\ldots,r_{\lambda_k})$. Then $n_{i+1} = m_{i+1}$ and the contents of the counting registers and the stack is unchanged. The algebra registers are given by

$$b_{i+1,j} = \sigma(b_{i\lambda_1},\ldots,b_{i\lambda_k}) = \sigma(\phi a_{i\lambda_1},\ldots,\phi a_{i\lambda_k}) \quad \text{if} \quad j=\mu$$
$$= b_{ij} \qquad\qquad = \phi a_{ij} \qquad\qquad \text{if} \quad j\neq\mu$$

using the induction hypothesis; but in the computation of $\alpha(a,x)$, $a_{i+1,j} = \sigma(a_{i\lambda_1},\ldots,a_{i\lambda_k})$ if $j=\mu$ and since ϕ is
$= a_{ij}$ if $j\neq\mu$
a homomorphism we can substitute $\phi\sigma(a_{i\lambda_1},\ldots,a_{i\lambda_k}) = \sigma(\phi a_{i\lambda_1},\ldots,\phi a_{i\lambda_k})$ to get $b_{i+1,j} = \phi a_{i+1,j}$.

The cases of the other algebraic instructions and the counting instructions are equally straightforward to check and so we next consider two algebraic conditional instructions. Here the contents of the algebra registers and the stack are unchanged and we have to determine n_{i+1}.

Let m_i be if $r_\mu = r_\lambda$ then u else v.
Then $n_{i+1} = u$ if $\phi a_{i\mu} = \phi a_{i\lambda}$. In the computation of $\alpha(a,x)$,
$= v$ otherwise

$m_{i+1} = u$ if $a_{i\mu} = a_{i\lambda}$, but $\phi a_{i\mu} = \phi a_{i\lambda}$ iff $a_{i\mu} = a_{i\lambda}$ as ϕ is
$= v$ otherwise

injective; thus $n_{i+1} = m_{i+1}$.

Let m_i be if $R(r_{\lambda_1},\ldots,r_{\lambda_k})$ then u else v.
Now $n_{i+1} = u$ if $R(\phi a_{i\lambda_1},\ldots,\phi a_{i\lambda_k})$, using the induction hypothesis
$= v$ otherwise

but as ϕ is a relational embedding $R(\phi a_{i\lambda_1},\ldots,\phi a_{i\lambda_k})$ is true iff

$R(a_{i\lambda_1},\ldots,a_{i\lambda_k})$ is true and since $m_{i+1} = u$ if $R(a_{i\lambda_1},\ldots,a_{i\lambda_k})$
$= v$ otherwise

we have $n_{i+1} = m_{i+1}$.

Finally there are the stacking instructions of which we consider one.

Let m_i be restore $(r_o,\ldots,r_{l-1},r_{l+1},\ldots,r_p)$. Only the instruction number and numerical registers are unaffected: the algebra registers are given by $b_{i+1,j} = b_{ij} = \phi a_{ij}$ if $j=1$
$= b^i_{tj} = \phi a^i_{tj}$ if $j\neq 1$

where there are t vectors in the stack at the i-th step. And the stack registers are given by $b^{i+1}_{t-1,k} = b^i_{t-1,k}$. In the computation of $\alpha(a,x)$ the formulae are $a_{i+1,j} = a_{ij}$ if $j=1$
$= a^i_{tj}$ if $j\neq 1$

and $a^{i+1}_{t-1,k} = a^i_{t-1,k}$. Comparing and using the induction hypothesis we maintain the identity.

Now to see that 3.1 follows from this state discription identity argue thus: as ϕ is total $\phi\alpha(a,x)\downarrow$ iff $\alpha(a,x)\downarrow$ so there is a k such that $\alpha(a,x) = a_{ko}$ whence $\phi\alpha(a,x) = \phi a_{ko}$. By the identity, $b_{ko} = \phi a_{ko}$ and so $D_k(\alpha,\phi a,x)$ is the last state description and $\phi\alpha(a,x) = \alpha(a,x)$. Similarly if a numerical output is obtained. Q.E.D.

3.2 COROLLARY <u>Let A and B be relational systems isomorphic under ϕ. Then the fapCS computation theories on A and B with code set ω are isomorphic under (id,ϕ).</u>

<u>Proof</u>: This follows from the fact the universal functions and step functions are fapCS-computable: for $e \in C$, $(a,x) \in A^n \times \omega^m$, ${}^{n,m}U(e,\phi a,x) \simeq \phi^{n,m}U(e,a,x)$ and ${}^{n,m}S(e,\phi a,x) \simeq {}^{n,m}S(e,a,x)$. Q.E.D.

This uniqueness theorem is pertinent when assessing the significance of our concept of computable function for it is most important that a formalised notion of constructiveness in algebraic systems be an abstract isomorphism invariant and 3.2 explicates a sense in which this is true of finite algorithmic procedures. About this technical requirement we find many writers in error - for example, G. Kreisel [16], pp 176-177 - and it is more carefully discussed in [27].

3.3 COROLLARY Each fapCS-semicomputable set $S \subset A^n$ is a characteristic subset of A.

Proof: Let α be a fapCS such that $S = \mathrm{dom}(\alpha)$, and let $\phi \in \mathrm{Aut}(A)$. For $a \in A^n$

$$\begin{aligned} \phi a \in S \quad &\text{iff } \alpha(\phi a)\downarrow \quad \text{by definition of } \alpha; \\ &\text{iff } \phi\alpha(a)\downarrow \quad \text{by 3.1;} \\ &\text{iff } \alpha(a)\downarrow \quad \text{as } \phi \text{ is total;} \end{aligned}$$

thus, $\phi : S \to S$. Q.E.D.

So, for example, in a group G any fapCS-semicomputable complex $S \subset G$ must be a normal complex, and, in particular, any subgroup which is not normal cannot be fapCS-semicomputable.

3.4 COROLLARY The symmetry group of a fapCS-computable valuation $v : A \to \omega$ or bihomomorphic form $\theta : A^n \to A$ is $\mathrm{Aut}(A)$.

Let $\mathrm{Emb}(A)$ be the semigroup of embeddings $A \to A$. 3.1 entails the centraliser of $\mathrm{Emb}(A)$ in the semigroup of all maps $A \to A$ contains ${}^1\mathrm{FAPCS}(A)$; if $\mathrm{Aut}_{\mathrm{fapCS}}(A)$ is the group of all fapCS-computable automorphisms of A and $Z(G)$ denotes the centre of group G then

3.5 COROLLARY $\mathrm{Aut}_{\mathrm{fapCS}}(A) < Z(\mathrm{Aut}(A))$ and thus $\mathrm{Aut}_{\mathrm{fapCS}}(A) \lhd \mathrm{Aut}(A)$.

This is not the place to enter a serious investigation of symmetry groups of general or particular algebras, we close this section with observations of automorphisms of groups which suggest the constructive implications of commutativity hypotheses.

3.6 PROPOSITION If G is a group with trivial centre then $\mathrm{Aut}_{\mathrm{fapCS}}(G)$ is trivial.

Proof: This follows from 3.5 and the fact that if a group G has trivial centre then Aut(G) has trivial centre, Kurosh [17] p.89. Q.E.D.

For example, any non-abelian simple group or any free group of rank > 1 has trivial centre. On the other hand, consider the power map $\exp(n,g) = g^n$ in the group G and which is a simple family of fapC-computable functions. If G is abelian then for each n, $\exp_n : G \to G$ is a homomorphism; when is this map a family of automorphisms? It is easy to verify that $\exp_n$ is injective for every n iff G is torsion-free, and that $\exp_n$ is surjective for every n iff G is divisible. Thus if G is a torsion-free divisible abelian group, $\mathrm{Aut}_{\mathrm{fapCS}}(G)$ contains the infinite fapCS-computable family $\exp(n,g)$.

4. Subalgebras and their enumeration

This section is about constructive enumerations in algebraic systems but contains results of a specifically algebraic and computation-theoretic nature in equal measure. This first theorem is quite basic and its first two corollaries go some way to explain the computational commitment of the fapCS, see [27].

4.1 THEOREM There is a fapCS-computable enumeration of the finitely generated subalgebras of A: there is a uniformly fapCS-computable family of functions ${}^{n}L : A^{n} \times \omega \to A$ such that for each $a \in A^{n}$, ${}^{n}L(a) : \omega \to \langle a \rangle$ is a surjection; moreover, ${}^{n}L$ can be chosen so as to list the subalgebras in increasing order of term height.

Proof. Choose any uniformly recursive enumeration of the code sets for the term algebras ${}^{n}f : \omega \to {}^{n}\Omega$ and define ${}^{n}L(a,k) = {}^{n}TE({}^{n}f(k),a)$. To obtain on $\langle a \rangle$ an ordering according to term height choose the special enumeration of lemma 1.3. In both cases it is easy to check that a code for ${}^{n}L$ can be recursively obtained from codes for ${}^{n}TE$ and ${}^{n}f$ which are simply functions of n. Q.E.D.

4.2 COROLLARY There is a fapCS-computable enumeration of the sets of all finite sequences of elements from the finitely generated subalgebras of A: there is a uniformly fapCS-computable family of functions ${}^{n,m}L_{*} : A^{n} \times \omega \to A^{m}$ such that for each $a \in A^{n}$, ${}^{n,m}L(a) : \omega \to \langle a \rangle^{m}$ is a surjection.

This is straight forward; the property established in 4.2 we shall refer to as fapCS-local pairing.

4.3 COROLLARY There is a fapCS-computable local search operator for the fapCS-semicomputable relations: there is a uniformly fapCS-computable family of functions ${}^{n,m,k}v : A^{n} \times \omega^{m} \to A^{k}$ such that if $S \subset A^{n} \times \omega^{m} \times A^{k}$ is fapCS-semicomputable and there is $y \in \langle a \rangle^{k}$ such that $S(a,x,y)$ then ${}^{n,m,k}v(a,x)\downarrow$ and $S(a,x,{}^{n,m,k}v(a,x))$.

Proof. To be more precise about the nature of the uniformity, define

$$ {}^{n,m,k}v(e,a,x) = {}^{1}TE^{k}(\delta z.Step(e,a,x,{}^{n,m}L_{*}(a,z)) = 0) $$

which is fapCS-computable by 2.9. Q.E.D.

The numerical search of 2.7, used in this argument, together with the property in 4.3 we shall refer to as fapCS-local search. 4.2 and 4.3 has this corollary:

4.4 COROLLARY If A is finitely generated then for each set of generators $a \in A^n$ there is uniform global pairing and a uniform global search operator fapCS-computable over (A,a); in particular prime algebras have fapCS-global pairing and fapCS-global search.

4.5 LEMMA There is a uniformly fapCS-computable family of separating functions ${}^{n,m}d : A^{n+m} \to A$ such that for each $a \in A^n$, $b = (b_1,\ldots,b_m) \in A^m$ if $\exists y \in \langle a \rangle . \bigwedge_{i=1}^{m} y \neq b_i$ then ${}^{n,m}d(a,b) \in \langle a \rangle$ and ${}^{n,m}d(a,b) \neq b_i$ for $1 \leq i \leq m$.

4.6 COROLLARY The membership relation for finitely generated subalgebras of A, ${}^{n}M(b,a) \equiv b \in \langle a \rangle$ for $b \in A$, $a \in A^n$, is uniformly fapCS-semicomputable.

Proof. ${}^{n}M(b,a)$ iff $\exists k .\ {}^{n}L(a,k) = b$. The function ${}^{n}f(b,a) = (\mu k \in \omega).{}^{n}L(a,k) = b$ is fapCS-computable by 2.7 and has domain ${}^{n}M$; uniformity of specification is immediate. Q.E.D.

In section nine we shall show that ${}^{1}M$ is not in general fapCS-decidable in abelian groups.

4.7 THEOREM There is a fapCS-computable enumeration ${}^{n}L_o$ of the finitely generated subalgebras of A wherein each subalgebra is enumerated without repetions and in increasing order of element height: for $a \in A^n$, ${}^{n}L_o(a) : \omega \to \langle a \rangle$ is a bijection and if $i \leq j$ then $Ht({}^{n}L_o(a)(i)) \leq Ht({}^{n}L_o(a)(j))$.

Proof. The idea of the algorithm is to renumber a list of all the elements of $\langle a \rangle$ in increasing order of height, such as that L mentioned in 4.1, evaluating its c-th element to discover whether

or not it has been obtained previously, as an element of less or equal height, by running L as far as the $(c-1)$-th element. If it has already appeared then one considers the $(c+1)$-th element, if not one adds 1 to a counter m so that then the c-th element of L is the m-th new element of the subalgebra; when $m=k$ one has $L_o(a)(k)$.

This is implemented by the following fapCS which has $r_1,..,r_n,c_1$ as input registers, r_o as output and numerical and algebraic working registers m,c,i and r,r'. m counts the number of distinct elements to be found in the first c elements of L and r holds the c-th element as it is compared against the i-th element previously evaluated and contained in r'.

1. $m := 0$
2. $c := 0$
3. if $m=c_1$ then 13 else 4
4. $c := c+1$
5. $r := L(r_1,\ldots,r_n,c)$
6. $i := 0$
7. if $i=c-1$ then 11 else 8
8. $i := i+1$
9. $r' := L(r_1,\ldots,r_n,i)$
10. if $r=r'$ then 4 else 7
11. $m := m+1$
12. goto 1
13. $r_o := r$, H.

L_o lists the elements in increasing order of height automatically from L; that the algorithm is uniform in n is easy to see for a code for ${}^nL_o$ can be recursively constructed from that of ${}^nL$. Q.E.D.

4.8 COROLLARY There is a fapCS-computable enumeration ${}^{n,m}L_o^*$ without repetitions of the sets of all finite sequences of elements from the finitely generated subalgebras of A: for $a \in A^n$, ${}^{n,m}L_o^*(a)$: $\omega \to \langle a \rangle^m$ is a bijection.

These next two consequences of 4.7 are essential to the success of any proposed method of computing in Algebra.

4.9 COROLLARY *The growth function* ${}^{n}g : A^{n} \times \omega \to \omega$ *defined* ${}^{n}g(a,k)$ = *the number of elements of* $\langle a \rangle$ *of height* $\leq k$ *is uniformly fapCS-computable.*

Proof. Define ${}^{n}g(a,k) = (\mu z \in \omega)(Ht(L(a,z)) = k \;\&\; Ht(L(a,z+1)) = k+1)$ where L is the enumeration of 4.7; this is fapCS-computable because Ht is from 2.8. Q.E.D.

4.10 COROLLARY *The order function* ${}^{n}ord : A^{n} \to \omega$ *defined* ${}^{n}ord(a) = |\langle a \rangle|$ *is uniformly fapCS-computable.*

Proof. Define ${}^{n}ord(a) = (\mu z \in \omega)({}^{n}g(a,z) \simeq {}^{n}g(a,z+1))$. Q.E.D.

It is important to notice that if A has fapC-computable term evaluation then all our programmes are fapC's. My experience suggests that all computations an algebraist might care to make will be *fapC-computable from term evaluation* at once fixing the necessity and influence of the stacking mechanism.

We conclude this section with two results about complexity measures, suggested by the paper of Blum [3], which illustrate the use of the enumeration theorems and also show how local theorems arise in the Computation Theory of our fapCS-computable functions. (Here we must leave to the reader the task of restoring certain details in our arguments which have been omitted, relying on his or her knowledge of ordinary recursion theory.)

4.11 THEOREM *Let* Φ *be fapCS-complexity measure. There are fapCS-computable functions* $\phi : C \times A^{m} \to C$ *and* $f : A^{m} \to \omega$ *such that for each* $e \in C$, $a \in A^{m}$, *with* $\{e\}$ *total on* $\langle a \rangle^{n}$ *and* $\langle a \rangle$ *finite,* $\{\phi(e,a)\} = \{e\}$ *over* $\langle a \rangle^{n}$ *and* $\Phi(\phi(e,a),b) \leq f(a)$ *for all* $b \in \langle a \rangle^{n}$. *Moreover, the family is uniformly fapCS-computable from* n,m *and a code for* Φ.

Proof. This is organised around two lemmas:

4.12 LEMMA *There is a uniform fapCS-computable enumeration* $^{n,m}\Psi : A^m \times \omega \times A^n \to A$ *of the n-ary functions totally defined on the m-generator finite subalgebras of* A.

Proof. This comes from uniformly enumerating the graphs of the finite n-ary functions over the finite subalgebras of A and numerically coding them by means of pairing on ω. Beginning with L_* and L of 4.2 and 4.1 which together give the enumeration $L_* \times L : \omega^2 \to \langle a\rangle^{n+1}$ we can obtain the graphs by coding as follows: Let $N = N(a) = |\langle a\rangle|^n$. The sequence numbers $\{\langle(1,k_1),\ldots,(N,k_N)\rangle \in \omega^{2N} : 1 \leq k_i \leq |\langle a\rangle| \ \&\ 1 \leq i \leq N\}$ are listed without repetition and so that $k = \langle(1,k_1),\ldots,(N,k_N)\rangle$ is interpreted as labelling the k-th map $\psi_{a,k}$ defined as $L_*(a,i) \mapsto L(a,k_i)$. Let u be the fapCS-computable unpacking function $u(a,k,i) = k_i$, $1 \leq i \leq N$. Our enumeration is

$$\psi_{a,k}(b) = {}^{n,m}\Psi(a,k,b) = L(a,u(a,k,(\mu z \in \omega)(L_*(a,z) = b)))$$

which is uniformly constructed from n,m. Q.E.D.

Now let $\psi(a,k)$ denote the official fapCS-code of the k-th function $\psi_{a,k}$. ψ is a fapCS-computable function because a code for $\psi_{a,k}$ can be obtained from a,k and codes for L_* and L by the uniformity of composition, the μ-operator and the Iteration Property.

4.13 LEMMA *There is a fapCS-computable* $f : A^m \to \omega$ *such that for every* $k \in \omega$, $b \in A^n$, $a \in A^m$ *with* $\langle a\rangle$ *finite,* $\Phi(\psi(a,k),b) \leq f(a)$.

Proof. To see that this is true one has to inspect all the component calculations of $\Psi(a,k,b)$ and show their Φ-complexity is bounded by a number fapCS-computable from a, actually the computations turn out to be recursive functions of $|\langle a\rangle|$. We begin by

observing that the Φ-cost of enumerating $\langle a\rangle$ can be bounded by a function $g(a) \approx \sum_{i=1}^{|a|} \Phi(\hat{L},i,a)$, where $\hat{L}$ is a code for L, and hence that of the enumeration of $\langle a\rangle^n$ by another function say $g(n,a)$. Now one shows the Φ-cost of searching this enumeration as far as the $N(a)$-th element and computing the appropriate components $u(a,k,i)$ can be bounded by computable functions say $W(a)$, $U(a)$. For example, $W(a) \approx \sum_{i,j=1}^{N(a)} \Phi(e_o,i,j,a)$ where e_o is a code for the fapCS-computable relation $L_*(a,i) = L_*(a,j)$. From these bounding functions and composition rule of adding complexity values a suitable f can be constructed. Q.E.D.

Thus if e codes a fapCS-computable n-ary function total on $\langle a\rangle^n$ then there is a k such that $\{e\}(b) \simeq \{\psi(a,k)\}(b)$ and $\Phi(\psi(a,k),b) \leq f(a)$ for all $b \in \langle a\rangle^n$. To complete 4.11 we have to show there is a fapCS-computable function $h : C \times A^m \to \omega$ which calculates the k from e,a for then $\phi(e,a) = \psi(a,h(a,e))$. Define $k(i) = (\mu z \in \omega)(L(a,z) = {}^nU(e,L_*(i,a))$ and then $h(e,a) = \langle(1,k(1)),\ldots,(N(a),kN(a))\rangle$. Q.E.D.

Actually 4.11 is included to emphasise this next fact which is not so obvious:

4.14 THEOREM Let Φ be a fapCS-complexity measure. There is a fapCS-computable function $\phi : C \times A^m \to C$ such that for each $e \in C$, $a \in A^m$ if $\{e\}$ is total on $\langle a\rangle^n$ and $\langle a\rangle$ is infinite then for any $\hat{f} \in C$ such that $\{\hat{f}\} = \{\phi(e,a)\}$ on $\langle a\rangle^n$ the set $Z_{\hat{f}} = \{b \in \langle a\rangle^n : \Phi(\hat{f},b) \leq \Phi(e,b)\}$ is finite.

More informally expressed, given e,a with $\langle a\rangle$ infinite and $\{e\}$ total on $\langle a\rangle^n$ we can effectively prescribe a function by code $\phi(e,a)$ such that all programmes which compute it are strictly more Φ-complex than e almost everywhere over $\langle a\rangle^n$.

Proof. From 4.8, let $L_* : A^m \times \omega \to A^n$ be a fapCS listing of the n-fold products of the m-generator algebras without repetitions. Given e,a we construct in stages over $L_*(a,k)$ a function f which is total on $\langle a \rangle^n$. For the moment fix a and let $\alpha(k) = L_*(a,k)$.

To compute $f\alpha(k)$, first decide the following k relations

$$\Phi(j,\alpha(k)) \leq \Phi(e,\alpha(k)) \quad 1 \leq j \leq k$$

and store those j for which the inequality is true: this procedure is fapCS-computable when $\{e\}$ is a total function as the right hand side is then always defined, and because we have pairing on ω. Notice that each programme code stored defines a total function. Now define $f\alpha(k) \neq$ value of any of these stored programmes applied to $\alpha(k)$, this can be fapCS-executed by means of 4.5 provided $\langle a \rangle$ is infinite:

$$f\alpha(k) = {}^{n,t}d(a,U(j_1,\alpha(k)),\ldots,U(j_t,\alpha(k)))$$

where $t = t(a,k)$ is the number of programmes stored and is fapCS-computable. Given e,a we can certainly fapCS-compute a code for f so defined by (say) $\phi(e,a)$. We have to prove f is sufficiently complex.

Let $\bar{Z}_{\hat{f}} = \{k \in \omega : \Phi(\hat{f},\alpha(k)) \leq \Phi(e,\alpha(k))\}$ and assume $\hat{f}$ computes f. We claim $\bar{Z}_{\hat{f}}$ is finite.

Let $k_o \in \omega$. If $k_o \geq \hat{f}$ then at stage k_o when defining $f\alpha(k_o)$, $\Phi(\hat{f},\alpha(k_o))$ was computed. If this computation had Φ-cost $\leq \Phi(e,\alpha(k_o))$ then $f\alpha(k_o)$ was made different from $\{\hat{f}\}(\alpha(k_o))$. But since these values are not distinct $k_o \notin \bar{Z}_{\hat{f}}$. Thus $k_o \in \bar{Z}_{\hat{f}}$ implies $k_o < \hat{f}$ and so $\bar{Z}_{\hat{f}}$ is finite. Q.E.D.

5. Semicomputable sets

In section one we adopted, without hesitation, the definition that the fapCS-semicomputable sets are those which arise as the domains of fapCS-computable functions. But there are other plausible definitions according to our experience with the several equivalent characterisations of the recursively enumerable subsets of ω. Here are recorded five ideas about enumeration, three of which we develop just as far as is necessary to compare the general algebraic situation with that in arithmetic, in fact we learn the reason for the stability of the concept there.

A set $S \subset A^n \times \omega^m$ is the numerical projection of a fapCS-computable relation or more simply, fapCS-numerically projectible if there is a fapCS-computable relation $C \subset A^n \times \omega^{m+1}$ such that $(a,x) \in S$ iff $\exists y.C(a,x,y)$.

5.1 PROPOSITION *S is fapCS-semicomputable if, and only if, S is fapCS-numerically projectible.*

Proof. If S is the projection of C define $f(a,x) \simeq (\mu z \in \omega).C(a,x,z)$: f is fapCS-computable and obviously, $\exists y.C(a,x,y)$ iff $f(a,x)\downarrow$. Conversely, let e code a fapCS with domain S. By the step counting theorem 2.5, $(a,x) \in S$ iff $\exists y.\mathrm{Step}(e,a,x,y)$ presents S as a numerical projection of a fapCS-computable relation. Q.E.D.

5.2 PROPOSITION *If R and S are fapCS-semicomputable relations then R & S and R v S are fapCS-semicomputable.*

Proof. First observe that if R and S are fapCS-computable then so are $R \,\&\, S$ and $R \vee S$. For example, consider $R \vee S$ whose characteristic function is computed by this fapCS with input

registers $r_1,\ldots,r_n,c_1,\ldots,c_m$, output c_o and working registers c,c'.

1. $c := S(r_1,\ldots,r_n,c_1,\ldots,c_m)$
2. if $c=0$ then 6 else 3
3. $c' := R(r_1,\ldots,r_n,c_1,\ldots,c_m)$
4. if $c' = 0$ then 6 else 5
5. $c_o := 1$, H
6. $c_o := 0$, H.

If R and S are fapCS-semicomputable then, using 5.1, set $R(a,x) \equiv \exists y.C(a,x,y)$ and $S(a,x) \equiv \exists y.D(a,x,y)$. Now $(R\&S)(a,x) \equiv \exists y.C(a,x,y) \,\&\, \exists y.D(a,x,y)$ and using recursive pairing this can be written $\exists \langle y_1,y_2\rangle .C(a,x,y_1) \,\&\, D(a,x,y_2)$ which is a fapCS-numerical projection. And also $(R\vee S)(a,x) \equiv \exists y.C(a,x,y) \vee \exists y.D(a,x,y) \equiv \exists y.C(a,x,y) \vee D(a,x,y)$, a fapCS-numerical projection. Q.E.D.

Whence the intersection and union of semicomputable sets are semicomputable and actually uniform though we have made no effort to make this clear.

Now we turn to those sets which are the algebraic projections of fapCS-computable sets. $S \subset A^n \times \omega^m$ is <u>fapCS-algebraically projectible or Σ_1-enumerable</u> if there is a fapCS-computable relation $C \subset A^n \times \omega^m \times A$ such that $(a,x) \in S$ iff $\exists y.C(a,x,y)$.

A relation $S \subset A^n \times \omega^m \times (A/\omega)^k$ has <u>k-ary algebraic/numerical selection</u> if there is a selection function $\nu : A^n \times \omega^m \to (A/\omega)^k$ such that if $\exists y.S(a,x,y)$ then $\nu(a,x)\downarrow$ and $S(a,x,\nu(a,x))$ holds.

5.3 PROPOSITION <u>Let $S \subset A^n \times \omega^m \times A^k$ be a fapCS-semicomputable relation with a fapCS-computable selection function. Then $\exists y.S(a,x,y)$ is fapCS-semicomputable</u>.

<u>Proof</u>. Let S be defined by f and have selection function ν. It is easy to check that $\exists y.S(a,x,y)$ iff $f(a,x,\nu(a,x))\downarrow$. Notice that the projection operation is uniform in the codes for f and ν. Q.E.D.

Such selection functions are not widely available. For example, let R be a ring in which there are at most 2 square roots of any element, and let $C(a,y)$ be $y^2=a$. Selection functions for C are those normally written $\nu(a) = \pm\sqrt{a}$, and if one such ν was fapCS-computable then $\nu(a) \in \langle a \rangle$, by 1.1. Look at $R = \mathbb{Z}(\sqrt{2})$: $\nu(2) = \pm\sqrt{2}$ but, inside R, $\langle 2 \rangle$ is $\mathbb{Z}$ and so the functions ν cannot be fapCS-computable: whilst C is fapCS-computable we cannot deduce that its projection, the set of all elements having a square root in R, is fapCS-semicomputable. With 4.3 in mind, the existence of selection functions is explained as follows.

A relation $S \subset A^n \times \omega^m \times A^k$ is <u>locally k-ary sufficient</u> if for any $(a,x) \in A^n \times \omega^m$, if $\exists y \in A^k . S(a,x,y)$ then $\exists y \in \langle a \rangle^k . S(a,x,y)$.

5.4 PROPOSITION <u>Let S be fapCS-semicomputable. If $S \subset A^n \times \omega^m \times \omega^k$ then S has fapCS-computable k-ary numerical selection. Otherwise, if $S \subset A^n \times \omega^m \times A^k$ then S has fapCS-computable k-ary algebraic selection if, and only if, S is locally k-ary sufficient</u>.

<u>Proof</u>. The first statement follows from the use of the μ operator on ω, proposition 2.7. Let $S \subset A^n \times \omega^m \times A^k$ be fapCS-semicomputable and have fapCS-computable selection function ν : given $(a,x) \in A^n \times \omega^m$, if $y \in A^k$. $S(a,x,y)$ then $\nu(a,x)\downarrow$ and $S(a,x,\nu(a,x))$. By applying the subalgebra theorem 1.1 to the component functions of ν puts $\nu(a,x) \in \langle a \rangle^k$, that is, S is locally sufficient. Conversely, the local search operators of 4.3 provide <u>full</u> selection for locally sufficient relations. Q.E.D.

In the light of this consider the fapCS-semicomputable sets in the large. With 4.4 in mind, the fapCS-semicomputable relations on

A will be said to have uniform, global fapCS-selection if for each n,m,k there is a fapCS-computable function $\nu : \omega \times A^n \times \omega^m \to (A/\omega)^k$ such that for any appropriate fapCS code e, if $\exists y.\{e\}(a,x,y)\downarrow$ then $\nu(e,a,x)\downarrow$ and $\{e\}(a,x,\nu(e,a,x))\downarrow$, the code for ν being uniformly computable from n,m,k. Inspecting the argument of 5.4 shows us

5.5 PROPOSITION The fapCS-semicomputable relations on A have uniform global fapCS-selection if, and only if, every fapCS-semicomputable relation is locally sufficient.

We can improve upon this characterisation.

A relational system A will be said to be locally algebraically complete if for each $n > 0$, any $a \in A^n$ and any finite family of polynomial vectors $\{p_i = (p_1^i,\ldots,p_{k_i}^i) : 1 \leq i \leq m\}$ from $T[X_1,\ldots,X_{n+1}]$, if $\exists y. \bigwedge_{i=1}^m S_i(p(a,y))$ then $\exists y \in \langle a \rangle \bigwedge_{i=1}^m S_i(p(a,y))$ wherein S_i is a basic relation of A or its negation, or is the equality or inequality relation, of appropriate rank k_i.

5.6 THEOREM The fapCS-semicomputable relations on A have uniform global fapCS-selection if, and only if, A is locally algebraically complete.

Proof. Given 5.5, we have to prove that every fapCS-semicomputable relation over A is subalgebra sufficient iff A is locally algebraically complete. One direction is automatic because local algebraic completeness asks only that certain polynomially definable (and so semicomputable) relations be subalgebra sufficient. Assume, then, that A is locally algebraically complete and that $S \in A^n \times \omega^m \times A^k$ is fapCS-semicomputable, the domain of fapCS α; we have to show that if $\exists y.\alpha(a,x,y)\downarrow$ then $\exists y \in \langle a \rangle.\alpha(a,x,y)\downarrow$. Let

$\exists y.\alpha(a,x,y)\downarrow$ and consider the computation $\alpha(a,x,y)$. If no algebraic conditional instructions appear in the unfolding of $\alpha(a,x,y)$ then α computes a polynomial, this being total makes $S = A^n \times \omega^m \times A^k$ and so subalgebra sufficient, the details of this observation are left as an exercise. Otherwise, make a list S_i of all the conditional relations which were used and found to be true, a list of basic relations and their negations. Applying the hypothesis of completeness there exists $y_o \in \langle a \rangle$ such that this list S_i of relations is still true of (a,x,y_o). This means that the computation $\alpha(a,x,y_o)$ is syntactically identical with that of $\alpha(a,x,y)$ - by another straightforward induction we prefer to omit - and so that $\exists y_o \in \langle a \rangle.S(a,x,y_o)$. Q.E.D.

5.7 COROLLARY <u>If A is locally algebraically complete then the fapCS-semicomputable sets are fapCS-uniformly closed under projection and, in particular, the Σ_1-enumerable sets are fapCS-semicomputable</u>.

Clearly, any prime algebra is locally algebraically complete as it has no local structure. As a converse to 5.7 we later prove 7.10 that <u>if A has standard fapS internal counting then every fapCS-semicomputable set is Σ_1-enumerable</u>, but it is best to place the conclusion of these efforts here:

5.8 THEOREM <u>If A is locally algebraically complete and has standard fapS internal counting, as such are many infinite prime algebras, then Σ_1-enumerability coincides with fapCS-semicomputability</u>.

We can make standard internal counting (which is explained in section seven, of course) widely available in every-day algebraic systems and so it is common to find fapCS-semicomputability as

simply a more stringent, less inclusive constructive concept that Σ_1-enumerability, rather than something incomporable. And Σ_1-computable sets arise naturally in the course of the simplest of algebraic calculations as in our ring example or in the integration example in section nine which are, in fact, diophantine sets over their algebras. Moreover they are theoretically prominent in that they are characteristic like the semicomputable sets in 3.3, they begin the abstract algebra analogue of the Kleene-Mostowski hierarchy over ω and so on; a deservedly more thorough account of them is to be included in [31], along with these last two ideas:

$S \subset A^n \times \omega^m$ is said to be fapCS-formally enumerable if there is a total fapCS-computable function $f : A \to A^n \times \omega^m$ such that $S = im(f)$. The idea is deficient in constructivity in much the same way that Σ_1-enumerability initially strikes us, actually if S is fapCS-formally enumerable then it is Σ_1-enumerable because $(a,x) \in S$ iff $\exists y.f(y) = (a,x)$.

More important is that $S \subset A^n \times \omega^m$ is fapCS-constructively enumerable if there is a fapCS-computable total function $f : \omega \to A^n \times \omega^m$ with $S = im(f)$. Such sets are fapCS-semicomputable for $g(a,x) = (\mu z \in \omega).f(z) = (a,x)$ is a fapCS-computable function with domain S. And to see that not all semicomputable sets need be constructively enumerable look at semicomputable sets which are not countable such as the elements of finite order in $GL(2,\mathbb{R})$.

6. The influence of the stack

In this and the next two sections we look at the mutual dependencies of stacking, counting, and their conjunction, in groups, fields and so forth. Recalling section two we have said (and here we shall show) that whilst in general the four kinds of fap computation are distinct (theorem 2.4) the everyday situation in Algebra has these inclusions

$$FAP(A) \longrightarrow FAPS(A) \longrightarrow FAPC(A) = FAPCS(A)$$

because term evaluation is usually fapC-computable. With this in mind define an algebra A to be regular iff A has uniformly fapC-computable term evaluation. We shall now try to determine when an algebraic structure is regular.

Stacking is essential because, in general, the fixed, finite number of algebra registers of the A-register machine with counting are inadequate in evaluating all terms. But one set of circumstances in which the unbounded storage of the stack can be avoided lie behind our tool which is this technical idea of Friedman (cf.[9], pp.376-377).

An n-ary syntactic development of width m is a sequence $T_1,\ldots,T_l$ such that

(i) each T_i is a list of m terms from $T[X_1,\ldots,X_n]$, $1\leq i\leq l$;

(ii) T_1 contains only indeterminates;

(iii) for $1\leq i\leq l-1$, either T_{i+1} arises from T_i by applying some k-ary operation symbol σ of A to some of the terms $t_1,\ldots,t_k$ in T_i and replacing one of the terms of T_i by $\sigma(t_1,\ldots,t_k)$ or,

(iv) T_{i+1} arises from T_i by replacing one of the terms of T_i by an indeterminate or by another of its terms;

(v) T_{i+1} differs from T_i in at most one of its terms $1\leq i\leq l$.

A term $t(X_1,\ldots,X_n)$ is of <u>width m</u> if it belongs to some n-ary syntactic development of width m. Obviously, $t(X_1,\ldots,X_n)$ is a term of width m iff it could appear as a term in a syntactic state description of a fap computation over A involving n inputs and m registers in all.

A (<u>fapC-computable</u>) <u>A-normal form of bounded width m(n)</u> is a set N of terms of width m(n) and a fapC-computable function $h : \omega \times \Omega \to \Omega$ over A such that for each n, $h(n) : {}^n\Omega \to {}^n\Omega$ and every i, $[h(i)] \in N$ and $[i] \equiv_A [h(i)]$, and $m : \omega \to \omega$ is fapC-computable.

6.1 THEOREM <u>Let A have a fapC-computable normal form of width m(n). Let M be the largest airity of the operations of A. Then there is a fapC involving (M+1)m(n) working registers which computes n-ary term evaluation. In particular, term evaluation is uniformly fapC-computable and the fapC and fapCS-computable functions coincide</u>.

<u>Proof</u>. The first assertion is essentially Friedman's lemma 1.6.2 and the other conclusions are immediate from our discussion here. Q.E.D.

Our definition of an A-normal form is, of course, an adaptation of the idea of the <u>varietal normal form</u> and which is a constructive concept, see H. Lausch & W. Nöbauer [<u>20</u>], p.23:

6.2 PROPOSITION <u>Let $\underline{V}$ be a variety. If the $\underline{V}$-free polynomial algebras $T_{\underline{V}}[X_1,\ldots,X_n]$ have a uniform varietal normal form of width m(n) then each algebra A of $\underline{V}$ has an A-normal form of width m(n)</u>.

<u>Proof</u>. That N is a varietal normal form means that given any $t \in T[X_1,\ldots,X_n]$ we can constructively find $t' \in N$ such that

$t = t'$ in $T_{\underline{V}}[X_1,\ldots,X_n]$, uniformly in n. By remark 1.2, for each $A \in \underline{V}$, $t \equiv_A t'$. Thus N is an A-normal form of width $m(n)$ for any A in $\underline{V}$. Q.E.D.

6.3 THEOREM *If the variety $\underline{V}$ has a varietal normal form of bounded width then for each algebra A in $\underline{V}$, FAPC(A) = FAPCS(A).*

We have mentioned before that groups, rings and fields are regular and it is not hard to prove this by writing fapC's for term evaluation. Excepting fields, which do not constitute a variety, 6.3 explicitly covers most of Algebra:

6.4 THEOREM *The varieties of semigroups, groups, associative rings, Lie rings, semilattices, lattices, Boolean algebras have varietal normal forms of bounded width.*

Proof. Varietal normal forms for these structures are well known; see Lausch & Nöbauer [20], P. Hall [13]. Given the varietal normal forms are constructive we have only to determine their widths, here is one case for illustration.

6.5 LEMMA *There is a varietal normal form of width 2 for groups.*

Proof. Consider the usual normal form for the free groups $N = \left\{ X_{\lambda_1}^{m_1} \ldots X_{\lambda_k}^{m_k} : k,\ \lambda_k \in \omega, m_i \in \mathbb{Z} \right\}$ with appropriate conditions on m_i; see [20], p.30. We check the width condition on N by induction on word length l. The basis $l=0$ is obvious. Assume all words in N of length $< l$ are of width 2 and consider w of length l. There are two cases $w = w_o X_\lambda$ and $w = w_o X_\lambda^{-1}$ of which we take just the first. Let $T_1,\ldots,T_k$ be an n-ary syntactic development of width 2 with $T_k = (w_o, w_1)$. Adding $(w_o, X_\lambda), (w_o X_\lambda, X_\lambda)$ we get a sequence of width 2 containing w. Q.E.D.

7. The influence of arithmetic

An algebra A is said to have fap, or fapS, counting on a constant $a \in A$ if there are functions $S,P : A \to A$ such that (i) $\{S^n a : n \in \omega\}$ is infinite, (ii) $PS^n a = S^{n-1} a$ for each n, and (iii) S,P are fap, or fapS, computable over (A,a). Obvious names for S,P are successor and predecessor. Notice (i) entails $S^n a = S^m a$ iff $n=m$.

7.1 PROPOSITION If A has fapS-counting on a constant a then $FAPS(A,a) = FAPCS(A,a)$ and if A has fap-counting on a then, in addition, $FAP(A,a) = FAPC(A,a)$. If A is regular and has fapS-counting on a constant a then $FAPS(A,a) = FAPC(A,a)$ and if A has fap-counting on a then, in addition, $FAP(A,a) = FAPCS(A,a)$.

Proof. Consider the first assertion. Let $f \in {}^nFAPCS(A,a)$ be computed by the fapCS α using algebra registers $r_0,\ldots,r_p$ and numerical registers $c_1,\ldots,c_q$. We must write a fapS β which will compute f over (A,a). Choose new algebra registers $u_1,\ldots,u_q$ and copy out the programme α with the following alterations.

Leave all algebraic instructions and stack instructions in α unchanged. Replace the counting instructions in α thus: the instruction in the left-hand column is exchanged for that in the right:

$c_\mu := 0$	$u_\mu := a$
$c_\mu := c_\lambda + 1$	$u_\mu := S(u_\mu)$
$c_\mu := c_\lambda - 1$	$u_\mu := P(u_\mu)$
if $c_\mu = c_\lambda$ then i else j	if $u_\mu = u_\lambda$ then i else j

To show the resulting programme β fapS-computes α over (A,a) one proves by induction on the length l of a computation $\beta(b)$, for

$b \in A^n$, the identity that if $D_i(\alpha,b) = (m_i;b_{ij},x_{ij},(z_{ij},b^i_{jk}))$ then $D_i(\beta,b) = (m_i;b_{ij},S^{x_{ij}}(a),(z_{ij},b^i_{jk}))$ for $1 \leq i \leq l$. This is ommitted.

The second assertion follows from this argument and the regular case statements are corollaries of the first two. Q.E.D.

7.2 PROPOSITION If A has fap or fapS counting on a constant then A is not locally 1-finite.

Proof. Let $a \in A$ be counting constant for successor function S. Then $\{S^n(a) : n \in \omega\}$ is infinite. By a trivial induction based upon theorem 1.1, $S^n(a) \in \langle a \rangle$ for each n, S is fap or fapS computable. Thus $\langle a \rangle$ is infinite. Q.E.D.

Consider some examples.

7.3 PROPOSITION A group G has fap counting on a constant if, and only if, G is not periodic.

Proof. The only if condition follows from 7.2 as periodicity and local 1-finiteness coincide in groups. Let G be non-periodic with $g \in G$ of infinite order. Define functions $S_g(x) = gx$ and $P_g(x) = g^{-1}x$ which are fap-computable over (G,g). Now $\{S^n_g(1) : n \in \omega\} = \{g^n : n \in \omega\}$ is infinite and $P_g S^n_g(1) = g^{-1}(g^n 1) = g^{n-1} = S^{n-1}_g(1)$. Thus G has fap counting on constant g. Q.E.D.

It is a surprising fact that A is not locally finite does not entail A has counting on a constant: take any of Golod's groups [11] which are periodic but not locally finite or, more spectacularly, one of Novikov and Adjan's m-generator (m>1) infinite periodic groups of finite exponent n, an odd number > 4381, see [2]. Still, this is not the case for fields:

7.4 PROPOSITION A field F has fap counting on a constant if, and only if, F is of characteristic 0 or F is of prime

characteristic but not algebraic over its prime subfield.

Proof. Let F have fap counting on a constant a and assume F is of characteristic p; denote the prime subfield of F by $\mathbb{Z}_p$. By the argument of 7.2, the subfield $\langle a \rangle = \mathbb{Z}_p(a)$ must be infinite and so cannot be algebraic over $\mathbb{Z}_p$ for otherwise $\mathbb{Z}_p(a)$ would be a finite field of order p^n where n is the degree of a over $\mathbb{Z}_p$.

Conversely, if F is of characteristic 0 then the additive group $(F,+)$ is not periodic by $1 \in F$ and so fap counting follows from 7.3. If F is of prime characterstic and contains an element t transcendental over its prime subfield then the multiplicative group $(F,\cdot)$ is not periodic by t and again fap counting follows from 7.3.

Q.E.D.

7.5 COROLLARY A field has fap counting on a constant if, and only if, it is not locally finite.

Proof. This follows from the fact that a field is locally finite if, and only if, it is of prime characteristic and is algebraic over its prime subfield. Q.E.D.

Noteworthy in the proof of 7.4 is the detail that in a field of characteristic 0 fap counting can be carried out on one of the basic constants, 1, of the structure: an algebra A is said to have fap, or fapS, internal counting if it has fap, or fapS, counting on a basic constant. From 7.1 we get

7.6 THEOREM If A has fapS internal counting then $FAPS(A) = FAPCS(A)$ and if it has fap internal counting then $FAP(A) = FAPC(A)$, in addition. If A is regular then A has fapS internal counting implies $FAPS(A) = FAPC(A)$ and A has fap internal counting implies $FAP(A) = FAPCS(A)$.

Any semiring R with 1 and of characteristic 0 has fap internal counting just as with a field, and since such algebras are regular FAP(R) = FAPCS(R). By a semiring we mean two semigroups joined by allowing one operation to distribute over the other: the obvious example is arithmetic $(\omega,+,-,1,0,=)$, and since ordinary recursion theory is the study of programmes on this semiring we have the last clause of 7.6 of an algebraic explanation why fap-computability is the minimal computation theory on ω.

A formal converse to 7.1 is this

7.7 PROPOSITION Let A be not 1-finite by $a \in A$. If FAPS(A,a) = FAPCS(A,a) then A has fapS counting on constant a; if FAP(A,a) = FAPCS(A,a) then A has fap counting on constant a. If A is regular then FAPS(A,a) = FAPC(A,a) implies A has fapS counting on a and FAP(A,a) = FAPC(A,a) implies A has fap counting on a.

Proof. Let FAPS(A,a)= FAPCS(A,a). We give functions on A which simulate counting and which are fapCS-computable over (A,a):

$$S_a(x) \simeq L_o(a,(\mu z\in\omega)(L_o(a,z) = x) + 1)$$
$$P_a(x) \simeq L_o(a,(\mu z\in\omega)(L_o(a,z) = x) - 1)$$

where L_o is the enumeration of theorem 4.7. Given x, S_a calculates the next element in the listing of $\langle a\rangle$, if $x \in \langle a\rangle$, and is undefined otherwise; and similarly P_a calculates the element before x, if $x \in \langle a\rangle$, and is undefined otherwise. Clearly these functions behave like successor and predecessor on $\langle a\rangle$. By hypothesis they are fapS computable over (A,a) and so constitute fapS counting machinery on constant a when we adduce the hypothesis that $\langle a\rangle$ is infinite. The other statements in 7.7 follow easily from this argument. Q.E.D.

Summarising, then:

7.8 THEOREM A has fap counting on a constant $a \in A$ if, and only if, A is not 1-finite by a and FAP(A,a) = FAPCS(A,a). If A is regular then A has fap counting on a if, and only if, A is not 1-finite by a and FAP(A,a) = FAPC(A,a).

Given the profusion of examples of structures where the various classes of computable functions coincide and, in particular, represent the minimal computation theory, the question arises When are some of these classes distinct? This is the topic of the next section. Lastly we prove what remains of 5.8.

The algebra A is said to have standard fap, or fapS, counting on constant $a \in A$ if A has fap, or fapS, counting on a and the characteristic function χ_a of $\langle a \rangle$ is fap, or fapS, computable.

7.9 PROPOSITION Let A have standard fapS counting on constant $a \in A$. Then if $S \subset A^n \times \omega^m$ is fapCS-semicomputable on A then S is Σ_1-enumerable over (A,a,χ_a).

Proof. By 5.1, S is the numerical projection of a fapCS-computable relation C, say $(b,x) \in S$ iff $(\exists y \in \omega)(C(a,x,y)=0)$. Let α be a fapCS defining S and let β be the standard transcription of α into an algebraic programme, a fapS, according to the rules of 7.1. As in 7.1 we can prove that if $\alpha(b,x,y)$ is computation of length l and $D_i(\alpha,b,x,y) = (m_i, b_{ij}, x_{ij}, (z_{ij}, b^i_{jk}))$ for $1 \leq i \leq l$, then $D_i(\beta,b,S^x(a),S^y(a)) = (m_i, b_{ij}, S^{x_{ij}}(a), z_{ij}, b^i_{jk}))$ for $1 \leq i \leq l$ and wherein if $x = (x_1,\ldots,x_m)$ then $S^x(a)$ denotes $(S^{x_1}(a),\ldots,S^{x_m}(a))$. This means that $\alpha(b,x,y) = 0$ iff $\beta(b,S^x(a),S^y(a)) = a$ and so $(\exists y \in \omega)(C(b,x,y) = 0)$ iff $(\exists y)(y \in \langle a \rangle \ \& \ \beta(b,S^x(a),S^y(a)) = a)$. Since

the bijection $i \to S^i(a)$ is fapCS-computable over (A,a,χ_a), because A has fapS counting, the theorem follows easily. Q.E.D.

In [31] we reveal that the hypothesis of standard fapS counting gives rise to a computation theory isomorphism between $FAPS(A_\omega)$ and $FAPS(A,a,\chi_a)$.

An algebra A has standard fapS internal counting if A has standard fapS counting on one of its basic constants.

7.10 COROLLARY If A has standard fapS internal counting then every fapCS-semicomputable set is Σ_1-enumerable.

From our examples following 7.6, many infinite prime algebras have such counting because there the appropriate characteristic functions are computable, see [31]. In section nine we learn that the membership relation for 1-generator subalgebras is not, in general, computable.

8. Locally finite algebras

8.1 THEOREM If A is locally n-finite then the halting problem for ${}^nFAPS(A)$ is fapCS-decidable: the relation ${}^nH(e,a)$ iff $\{e\}(a)\downarrow$ is fapCS-computable on $\Omega_S \times A^n$.

Proof. A state description in a fapS computation is a list of the form $(k;a_1,\ldots,a_p;(z_1;a_1,\ldots,a_{1p}),\ldots,(z_q;a_{q1},\ldots,a_{qp}))$. When A is n-finite the number of distinct state descriptions which may arise in a fapS computation with n inputs is bounded: let R(e) calculate the number of algebra registers used in programme e, I(e) calculate the number of instructions in e and M(e) calculate the number of markers for stack blocks in e, these are all recursive functions. If the state description above belonged to a

computation by e then $k \leq I(e)$, $p = R(e)$, and each $z_i \leq M(e)$. Thus there are at most $B_o(e,a) = ({}^n\mathrm{ord}(a)+1)^{R(e).(I(e)+1)}$ different states for the ordinary algebra registers and instruction number and due to the nature of stacking there can be at most $B_S(e,a) = (B_o(e,a).(M(e)+1))!$ different states for the stack. This means that the total number of distinct state descriptions available for $\{e\}(a)\downarrow$ is fapCS-computably bounded by $B(e,a) = B_o(e,a).B_S(e,a)$.

Claim 1. For $e \in \Omega_S, a \in A^n$, ${}^nH(e,a)$ iff ${}^nS(e,a) \leq B(e,a)$.

If ${}^nS(e,a) \leq B(e,a)$ then ${}^nS(e,a)\downarrow$ and so ${}^nH(e,a)$. Conversely, note that $\{e\}(a)\downarrow$ iff there are finitely many state descriptions the last of which has no element in the place of r_o or, there are infinitely many state descriptions to $\{e\}(a)$. In the first case $S^n(e,a)\uparrow$ so the right-hand-side condition is false, in the second case we know

Claim 2. There are infinitely many state descriptions to $\{e\}(a)$ iff there are two identical ordinary algebra state descriptions or two identical stack state descriptions.

For if either of two identical states D_i, D_j arise then the programme must regenerate after D_j exactly the states intermediate between D_i and D_j and so produce an infinite but periodic set of state descriptions. On the other hand, if there are infinitely many state descriptions in the unfolding of $\{e\}(a)$ then because the number of distinct states possible is finite there must arise repetitions, (Claim 2). Summing up we get claim 1: $\{e\}(a)\downarrow$ implies ${}^nS(e,a)\downarrow$ and, by claim 2, ${}^nS(e,a) \leq B(e,a)$. Now since ${}^nS$ is a complexity measure (theorem 2.7) and B is total on $\Omega_S \times A^n$, this relation of claim 1 is fapCS-decidable. Q.E.D.

Notice that if A is locally finite then the relation is uniformly fapCS-computable over all n. An algebra A will be said to be fapCS-formally valued or stratified if there is a total function $v : A \to \omega$ which is a fapCS-computable surjection; the sets $\{a : v(a) = n\} = v^{-1}(n)$ are the levels or strata of A under v. Clearly, a formal valuation v induces a prewellordering of A by $a \leq b$ iff $v(a) \leq v(b)$. (It is easy to prove that if A is not 1-finite then A is fapCS-stratifiable over A and a constant.)

8.2 THEOREM Let A be 1-finite and fapCS-formally valued by v. Then the set $K_v = \{a \in A : v(a) \in \Omega_S \ \& \ \{v(a)\}(a)\downarrow\}$ is fapCS-computable but not fapS-computable.

Proof. A 1-finite implies ${}^1H(e,a)$ is fapCS-decidable on $\Omega_S \times A$ by 8.1. Thus K_v is fapCS-decidable as $a \in K_v$ iff $v(a) \in \Omega_S \ \& \ {}^1H(v(a),a) = 0$, where v is total. Assume for a contradiction that K_v is fapS-computable. Then $\neg K_v$ is fapS-semicomputable and there exists a fapS α such that $\mathrm{dom}(\alpha) = \neg K_v$. Choose $b \in A$ such that $v(b)$ codes α. Then $b \in \neg K_v$ iff $\alpha(b)\downarrow$ by definition of α
iff $\{v(b)\}(b)\downarrow$ by definition of b,
iff $b \in K_v$.

Thus there is no such α and K_v is not fapS-computable. Q.E.D.

So if A is 1-finite and fapCS-formally valued then $\mathrm{FAPS}(A) \subset_{\neq} \mathrm{FAPCS}(A)$ and if in addition A is regular then $\mathrm{FAPS}(A) \subset_{\neq} \mathrm{FAPC}(A)$. One way to exhibit formally valued algebras is to use the order function ${}^1\mathrm{ord} : A \to \omega$ of 4.10. If A contains 1-generator subalgebras of every finite order then ${}^1\mathrm{ord}$ is fapCS-formal valuation, provided A is 1-finite. As an example, take

the locally finite group $\mathbb{Z}_\infty$ consisting of all the complex roots of unity.

More generally, let A be 1-finite and set $\pi(A) = \mathrm{im}({}^1\mathrm{ord})$, if $\pi(A)$ contains a recursive subset S then choose a recursive bijection $f : S \to \omega$ and define $v(a) = f\,{}^1\mathrm{ord}(a)$ if ${}^1\mathrm{ord}(a) \in S$; $= {}^1\mathrm{ord}(a)$ otherwise.

For an example among groups we must look for one of infinite exponent. Let p be a prime and let $\mathbb{Z}_{p^\infty}$ be the locally finite abelian group of all complex roots of unity which are of order some power of p. For these Prüfer groups, $\pi(\mathbb{Z}_{p^\infty}) = \{p^n : n \in \omega\}$, a recursive set.

Turning to fields let F be a locally finite field of characteristic p and observe that ${}^1\mathrm{ord}(a) = p^{d(a)}$ where $d : F \to \omega$ calculates the degree of $a \in F$, that is, $d(a) = \dim[\mathbb{Z}_p(a) : \mathbb{Z}_p]$. Since ${}^1\mathrm{ord}$ is fapCS-computable so is d and if F contains elements of every finite degree then d is a fapCS-formal valuation. Such an F can be chosen by taking the splitting field of the polynomials $\{X^{p^n} - X \in \mathbb{Z}_p[X] : n \in \omega\}$ but the best example is the algebraic closure K of $\mathbb{Z}_p$. Summarising,

8.3 THEOREM Over the groups $A = \mathbb{Z}_\infty, \mathbb{Z}_{p^\infty}$ and the field $A = K$,

$$\mathrm{FAPS}(A) \subsetneq \mathrm{FAPC}(A) = \mathrm{FAPCS}(A).$$

9. Topological algebras

A relational structure A is a topological algebraic system if the domain of the structure is a non-trivial topological space on which its operations are continuous: nothing is required of the basic relations belonging to A. A relation $R \in A^n \times \omega^m$ is said to be continuous at the point $(a,x) \in A^n \times \omega^m$ if its characteristic

function $R : A^n \times \omega^m \to \{0,1\}$ is continuous between the product of the given topology on A and the discreet topology on ω and the ciscreet topology on $\{0,1\}$.

The principal theorem of this last section tells about the topological qualities of the fapCS-computable subsets of A. It is suggested by perceptive remarks of Herman and Isard in [15] concerning $\mathbb{R}$. To state the result we have to define the notion of a transcendental point of A. Recall the equivalence relation $\equiv_A$ on $T[X_1,\ldots,X_n]$ defined in the first section: a point $a \in A^n$ is said to be transcendental if for any terms $t,t' \in T[X_1,\ldots,X_n]$, $t \equiv_A t'$ iff $v(t,a) = v(t',a)$ in A. Now $\equiv_A$ is a congruence relation on $T[X_1,\ldots,X_n]$ and the algebra $T_A[X_1,\ldots,X_n] = T[X_1,\ldots,X_n]/\equiv_A$ is appropriately called the n-ary equational core of A. Thus, $a \in A^n$ is transcendental iff $\langle a\rangle$ is isomorphic to $T_A[X_1,\ldots,X_n]$ by v_a.

9.1 LEMMA Let $\underline{V}$ be a variety and $A \in \underline{V}$. If $a \in A^n$ and $\langle a\rangle$ is a $\underline{V}$-free subalgebra of A then a is a transcendental point of A.

Proof. If $\langle a\rangle$ is $\underline{V}$-free then $v(a): T_{\underline{V}}[X_1,\ldots,X_n] \to A$ is an embedding: $t(a) = t'(a)$ in A iff $t = t'$ in $T_{\underline{V}}[X_1,\ldots,X_n]$. By remark 1.2, $t \equiv_A t'$ and a is transcendental. Q.E.D.

Here is our theorem.

9.2 THEOREM Let A be a Hausdorff topological algebraic system and let $S \subset A^n \times \omega^m$ be a relation which is fapCS-decidable from A. If A contains a transcendental point $a \in A^n$ on whose subalgebra $\langle a\rangle$ the basic relations of A are continuous then for any $x \in \omega^m$ there is an open subset of A^n containing a upon which S holds or fails accordingly as it holds or fails on a. In particular, for each $x \in \omega^m$, the sets $\{b \in A^n : S(b,x)\}$ and $\{b \in A^n : \neg S(b,x)\}$ cannot both be dense in A^n.

One can see immediately from 9.2 this important corollary:

9.3 COROLLARY Let A be a Hausdorff topological algebra without basic relations in the variety $\underline{V}$. If A contains a $\underline{V}$-free n-generator subalgebra then for any fapCS-decidable relation $S \subset A^n \times \omega^m$ and $x \in \omega^m$ the sets $\{a \in A^n : S(a,x)\}$ and $\{a \in A^n : \neg S(a,x)\}$ cannot both be dense.

Proof of 9.2. Let α fapCS decide S over A. Let $a \in A^n$ be transcendental and $x \in \omega^m$ be arbitrarily chosen. Since S is fapCS-decidable iff $\neg S$ is fapCS decidable we assume, without loss of generality, that S(a,x) is true and consider a computation $\alpha(a,x)$ of this fact. Let $\alpha(a,x)$ have length l and syntactic state descriptions $T_i(\alpha,a,x)$, for $1 \leq i \leq l$; we set $T_i(X) = T_i(\alpha,a,x)(X,x) = (m_i, t_{ij}(X), x_{ij}, (z_{ij}, t^i_{jk}(X)))$, where $X = (X_1, \ldots, X_n)$. And for $b \in A^n$, let $D_i(\alpha,b,x) = (n_i, b_{ij}, y_{ij}, (w_{ij}, b^i_{jk}))$. We shall prove there is a basic open set B(a) containing a and such that for all $b \in B(a)$ and each $1 \leq i \leq l$, $D_i(\alpha,b,x) = T_i(b)$. Obviously this entails $\alpha(b,x) = \alpha(a,x)$ for $b \in B(a)$.

Claim. For each $1 \leq i \leq l$, there is a basic open set $B_i(a)$ such that for each $b \in B_i(a)$, $T_{i+1}(b)$ succeeds $T_i(b)$ in the computation of $\alpha(b,x)$.

Whence our basic open set $B(a) = \bigcap_{i=1}^{l} B_i(a)$; we prove the claim by induction on i.

Now $D_1(\alpha,b,x) = (1, b_1, \ldots, b_n, \emptyset, \ldots, \emptyset, x_1, \ldots, x_m, \emptyset, \ldots, \emptyset)$ and $T_1(X) = (1, X_1, \ldots, X_n, \emptyset, \ldots, \emptyset, x_1, \ldots, x_m, \emptyset, \ldots, \emptyset)$ and obviously $D_1(\alpha,b,x) = T_1(b)$. To verify the claim is true for i=1 we have to compare how $D_2(\alpha,b,x)$ and $T_2(X)$ arise from $D_1(\alpha,b,x)$ and $T_1(X)$,

respectively. This depends on the nature of the first instruction: there are 12 cases but we shall give just two examples which anticipate the style of the general induction step.

Let 1 be $r_\mu := \sigma(r_{\lambda_1},\ldots,r_{\lambda_k})$. In applying this instruction only algebra registers are changed so $b_{2j} = \sigma(b_{1\lambda_1},\ldots,b_{1\lambda_k})$ if $j=\mu$
$= b_{1j}$ if $j\neq\mu$

and when 1 is applied to $T_1(X)$ we have $t_{2j}(X) = \sigma(X_{\lambda_1},\ldots,X_{\lambda_k})$ if $j=\mu$
$= X_j$ if $j\neq\mu$

Thus $t_{2j}(b) = b_{2j}$ for any $b \in A^n$: the operation is <u>independent</u> of a and we can take $B_i(a) = A^n$ for the claim to be true.

Let 1 be if $R(r_{\lambda_1},\ldots,r_{\lambda_k})$ then u else v. Here only the instruction number is changed:

$n_2 = u$ if $R(b_{1\lambda_1},\ldots,b_{1\lambda_k})$ and $m_2 = u$ if $R(a_{\lambda_1},\ldots,a_{\lambda_k})$. By the
$= v$ otherwise

continuity of R at a there exists a basic open set $V(a) \subset (R\circ p(X))^{-1}(0)$ where $p(X) = (X_{\lambda_1},\ldots,X_{\lambda_k})$. So for any $b \in V(a)$, $n_2 = m_2$; here the companion was <u>dependent</u> upon a.

Now assume as induction hypothesis that the open set $B_i(a)$ is constructed so that $b \in B_i(a)$ implies $D_i(\alpha,b,x) = T_i(b)$ and consider the passage from $D_i(\alpha,b,x)$ to $D_{i+1}(\alpha,b,x)$. First we deal with the operational instructions.

Let $n_i = m_i$ be $r_\mu := \sigma(r_{\lambda_1},\ldots,r_{\lambda_k})$. The algebra registers transform by these formulae:

$b_{i+1,j} = \sigma(b_{i\lambda_1},..,b_{i\lambda_k})$ if $j=\mu$ and $t_{i+1,j}(X) = \sigma(t_{i\lambda_1}(X),..,t_{i\lambda_k}(X))$ if $j=\mu$
$= b_{ij}$ if $j\neq\mu$ $= t_{ij}(X)$ if $j\neq\mu$

Substituting $X = b$ and applying the induction hypothesis we get $D_{i+1}(\alpha,b,x) = T_{i+1}(b)$ for any $b \in B_i(a) = B_{i+1}(a)$.

Thanks to the independence of the transformat other operational instruction cases are equally s omit them. Conditional transformations do depend

Let $n_i = m_i$ be if $R(r_{\lambda_1},\ldots,r_{\lambda_k})$ then u el the induction hypothesis $b_{ij} = t_{ij}(b)$, so

$n_{i+1} = u$ if $R(t_{i\lambda_1}(b),..,t_{i\lambda_k}(b))$ and $m_{i+1} = u$ if $R(t_{i\lambda_1}(a)$

$= v$ otherwise $= v$ otherwise.

By the continuity hypothesis on relations, the ma is continuous at a, where $p(X) = (t_{1\lambda_1}(X),\ldots,t_{i\lambda}$ is a basic open set $V_i(a)$ containing a such that $R \circ p(b) = R \circ p(a)$ and $n_{i+1} = m_{i+1}$. Thus define B_{i+}

Let $n_i = m_i$ be if $r_\mu = r_\lambda$ then u else v. Th $b_{ij} = t_{ij}(b)$ we have

$n_{i+1} = u$ if $t_{i\mu}(b) = t_{i\lambda}(b)$ and $m_{i+1} = u$ if $t_{i\mu}(a)$

$= v$ otherwise $= v$ otherwise

the two cases. If $t_{i\mu}(a) = t_{i\lambda}(a)$ then since a $t_{i\mu} \equiv_A t_{i\lambda}$ and $t_{i\mu}(b) = t_{i\lambda}(b)$ for any $b \in A^n$ and $T_i(X)$ to $T_{i+1}(X)$ is independent of a. Thus we s on which $n_{i+1} = m_{i+1}$.

However, if $t_{i\mu}(a) \neq t_{i\lambda}(a)$ then $\{b \in A^n : t_{i\mu}(b$ an open set containing a and we can choose a ne $V_i(a)$ about a within it (to do this A must be $b \in V_i(a)$, $n_{i+1} = m_{i+1}$ and here we set $B_{i+1}(a) = V$

9.4 COROLLARY Let A be a Hausdorff topolo $f : A^n \times \omega^m \to A/\omega$ a fapCS-computable total function an n+1/n-ary transcendental point on whose subalg relations of A are continuous then f is const neighbourhood of a.

To begin with, consider the field of real numbers $\mathbb{R}$ wherein any transcendental number is a transcendental element in our special sense. Therefore, by 9.2, sets such as the rationals $\mathbb{Q}$ and the algebraic numbers $\mathbb{A}$ which are dense and codense in $\mathbb{R}$ cannot be fapCS-decidable. To proceed to more complicated applications of 9.3 observe that a group has a transcendental element if, and only if, it has a free 1-generator subgroup, if, and only if, it has an element of infinite order. Here are two applications of 9.3 to abelian groups. Let T^n be the n-dimensional torus group, the n-fold direct product of the circle group S^1.

9.5 EXAMPLE The set $\mathrm{FinOrd}(T^n)$ of all elements of T^n of finite order, also known as the torsion subgroup of T^n, is fapCS-semicomputable but not fapCS-computable.

Proof. It is known from section one that $\mathrm{FinOrd}(T^n)$ is fapCS-semicomputable, we have to show T^n contains a 1-generator free-abelian subgroup and that $\mathrm{FinOrd}(T^n)$ is dense and codense in T^n. We begin by examining S^1.

Let $f : [0,1) \to S^1$ be the continuous surjection $f(t) = (\sin 2\pi t, \cos 2\pi t)$. Clearly, $f(t)$ is of finite order iff t is a rational number for $f(t)^n = (\sin 2\pi nt, \cos 2\pi nt) = (0,1)$ iff $2\pi nt$ is an integral multiple of 2π iff t is rational. Thus S^1 has a free-abelian subgroup and, indeed, since the rationals are dense and codense in $[0,1)$, $\mathrm{FinOrd}(S^1)$ is dense and codense in S^1 because the image of a dense subset of a space is dense in the image of a continuous mapping. Now observe that in any group G, $\mathrm{FinOrd}(G)^n = \mathrm{FinOrd}(G^n)$ and $\mathrm{InfOrd}(G)^n = \mathrm{InfOrd}(G^n)$ where $\mathrm{InfOrd}(G) = \neg\mathrm{FinOrd}(G)$. And also that in any toppological space X,

D dense in X entails D^n dense in X^n. Applying these facts to FinOrd(S^1) and its complement we get that FinOrd(T^n) is dense and codense and contains a 1-generator free-abelian subgroup. Q.E.D.

9.6 EXAMPLE The 1-generator subgroup membership relation M in T^n is fapCS-semicomputable but not fapCS-computable.

Proof. First we show $M = \{(g,t) : g \in \langle t \rangle\}$ is dense in $T^n \times T^n$. Let $B(a,b)$ be a basic open set containing $(a,b) \in T^n \times T^n$ which we can take, without loss of generality, to be of the form $B_1(a) \times B_2(b)$ where $B_1(a), B_2(b)$ are basic open sets about a,b respectively. Now the set of those $t \in T^n$ such that $\langle t \rangle$ is dense in T^n is itself dense in T^n, see J.F. Adams [1], p.79. So we can choose $t \in B_2(a)$ so that $\langle t \rangle$ meets all neighbourhoods and in particular $B_1(b)$. This means there is $(g,t) \in B_1(a) \times B_2(b)$ such that $g \in \langle t \rangle$ and M is dense.

Consider $\neg M$. Let $B(a,b)$ be as before. From the argument of 9.5 we can choose an element $t \in B_2(b)$ of finite order and $g \in B_1(a)$ of infinite order and so a pair $(g,t) \in B(a,b)$ for which $g \notin \langle t \rangle$.

From the observations of 9.5 it is easy to show T^n contains a 2-generator free abelian subgroup. Q.E.D.

Our last example applies 9.2 to the algebra of elementary integration. Let $C^\omega(\mathbb{R},\mathbb{R})$ be the set of all analytic functions $\mathbb{R} \to \mathbb{R}$ which is a differential ring under pointwise addition and multiplication, and differentiation. Let $\underline{E}$ be the differential subring generated by the polynomial functions $\mathbb{R}[X]$, e^x, $\sin x$ and all their compositions, a subring of the so-called elementary functions. Let I(f) be the integration relation in $\underline{E}$, I(f) iff $(\exists g \in \underline{E})(Dg=f)$. For example, it is well known from a theorem of Liouville that $e^{-x^2} \notin I$, see G.H. Hardy's book [14] about the relation I.

9.7 EXAMPLE I is a Σ_1-enumerable subset of $\underline{\underline{E}}$ which is not fapCS-computable.

Proof. Equip $C^\omega(\mathbb{R},\mathbb{R})$ with the C^∞-topology prescribed thus, a sequence $f_n \to 0$ as $n \to 0$ iff for each k, the sequence $D^k f_n \to 0$ as $n \to 0$ in the topology of uniform convergence on compact subsets on $C^\omega(\mathbb{R},\mathbb{R})$; this means that for each k and each real $R > 0$, the sequence $\sup_{|x|\leq R} |D^k f_n(x)| \to 0$ in $\mathbb{R}$. With the C^∞-topology $C^\omega(\mathbb{R},\mathbb{R})$ is a topological differential ring (which is not the case with the usual topology of uniform convergence on compacts where D fails to be continuous). See M. Golubitsky and V. Guillemin [12], pp.42-50. Consider the hypotheses of 9.2. By induction on term height it is straightforward to prove that (say) e^x is a transcendental point in $\underline{\underline{E}}$; this is ommitted. That I is dense in $\underline{\underline{E}}$ in the C^∞-topology follows from the fact that the polynomials $\mathbb{R}[X] < I < \underline{\underline{E}}$ and that the sequence of Taylor polynomials of an analytic function converge to the function in the topology of uniform convergence. That I is codense is more involved.

Let $f \in I$, we shall approximate f by the sequence of non-integrable functions $f_n(x) = f(x) + \frac{1}{n} e^{-x^2/n}$. It is easy to see that the g_n are not integrable and that the approximation property follows from the claim that $\frac{1}{n} e^{-x^2/n} \to 0$ as $n \to \infty$ in the C^∞-topology.

On calculating the k-th derivative $D^k(e^{-x^2/n})$ we find it to be of the form $P_k(x,1/n)e^{x^2/n}$, where $P_k(x,1/n) = \sum_{i=1}^{n} a_i X^i/n^{f_i(k)}$ where $f_i(k) \geq [k/2]$ = largest natural number $\geq k/2$. Whence it is easy to check that $D^k(e^{-x^2/n}) \to 0$ in the topology of uniform convergence on compact sets. Q.E.D.

I am grateful to B. Birkeland for his invaluable assistence with 9.7.

REFERENCES

1 J.F. ADAMS, Lectures on Lie groups (W.A. Benjamin, New York,1969).

2. S.I. ADJAN, The Burnside problem (Springer-Verlag, Berlin, 1979).

3 M. BLUM, 'A machine-independent theory of the complexity of recursive functions', J. Association Computing Machinery, 14 (1967) 322-336.

4 M. DAVIS, Computability, (Courant Institute Lecture Notes, New York, 1974).

5 Y.L. ERSHOV, 'Theorie der Numerierungen, I', Zeit. Mat. Logik Grund. Math., 19 (1973) 289-388.

6 ——— 'Theorie der Numerierungen, II', Zeit.Mat. Logik Grund. Math., 21 (1975) 473-584.

7 J.E. FENSTAD, 'Computation theories: an axiomatic approach to recursion on general structures', pp. 143-168 of G. MÜLLER, A. OBERSCHELP, and K. POTTHOFF (editors), Logic conference, Kiel 1974, (Springer-Verlag, Heidelberg, 1975).

8 ——— Recursion theory: an axiomatic approach, (Springer-Verlag, Berlin, 1980).

9 H. FRIEDMAN, 'Algorithmic procedures, generalised Turing algorithms, and elementary recursion theory', pp.316-389 of 10.

10 R.O. GANDY and C.E.M. YATES (editors), Logic Colloquium, '69, (North-Holland, Amsterdam, 1971).

11 E.S. GOLOD, 'On nil-algebras and residually finite p-groups'. Amer. Math. Soc. Translations, (2) 48 (1965) 103-106.

12 M. GOLUBITSKY and V. GUILLEMIN, Stable mappings and their singularities, (Springer-Verlag, New York, 1973).

13 P. HALL, 'Some word problems', J. London Math. Soc., 33 (1958) 482-496.

14 G.H. HARDY, The integration of functions of a single variable, Second edition, (Cambridge University Press, London, 1916).

15 G.T. HERMAN and S.D. ISARD, 'Computability over arbitrary fields', J. London Math. Soc., 2 (1970) 73-79.

16 G. KREISEL, 'Some reasons for generalising recursion theory', pp. 139-198 of 10.

17 A.G. KUROSH, The theory of groups I, (Chelsea, New York,1955)

18 ——— The theory of groups II, (Chelsea, New York, 1956).

19 ——— General algebra, (Chelsea, New York, 1963).

20 H. LAUSCH and W. NÖBAUER, Algebra of polynomials, (North-Holland, Amsterdam, 1973).

21 A.I. MAL'CEV, 'Constructive algebras, I', pp.148-212 of A.I. MAL'CEV The metamathematics of algebraic systems. Collected papers: 1936-1967, (North-Holland, Amsterdan, 1971).

22 ——— Algorithms and recursive functions, (Wolters-Noordhoff, Groningen,1970).

23 ——— Algebraic systems, (Springer-Verlag, Berlin, 1973).

24 J. MOLDESTAD, Computations in higher types (Springer-Verlag, Berlin, 1977).

25 J. MOLDESTAD, V. STOLTENBERG-HANSEN and J.V. TUCKER, 'Finite algorithmic procedures and inductive definability', Matematisk institutt, Universitetet i Oslo Preprint Series, No.6 (ISBN 82-553-0346-4), Oslo, 1978.

26 ——— 'Finite algorithmic procedures and computation theories', Matematisk institutt, Universitetet i Oslo Preprint Series, No. 7 (ISBN 82-553-0347-2), Oslo 1978.

27 J. MOLDESTAD and J.V. TUCKER, 'On the classification of computable functions in an abstract setting', Matematisk institutt, Universitetet i Oslo Preprint Series, No. 13 (ISBN 82-553-0359-6), Oslo, 1978.

28 Y.N. MOSCHOVAKIS, 'Axioms for computation theories - first draft', pp. 119-255 of 10.

29 J.C. SHEPHERDSON and H.E. STURGIS, 'Computability of recursive functions', J. Association Computing Machinery, 10 (1963) 217-255.

30 V. STOLTENBERG-HANSEN,'Finite injury arguments in infinite computation theories', Matematisk institutt, Universitetet i Oslo Preprint Series, No 12 (ISBN 82-553-0313-8), Oslo, 1977.

31 J.V. TUCKER,'Finite algorithmic procedures and first-order definability',in preparation.

32 B.L. VAN DER WAERDEN, Algebra I, (Ungar, New York, 1970).

UNIVERSITETET I OSLO
MATEMATISK INSTITUTT
POSTBOKS 1053
BLINDERN
OSLO, 3.